FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Anton Paatelma

802.11 PAYLOAD ITERATIVE DECODING BETWEEN MULTIPLE TRANSMISSION ATTEMPTS

Master's Thesis
Degree Programme in Electronics and Communications Engineering
December 2021

Paatelma A. (2021) 802.11 Payload Iterative Decoding between Multiple Transmission Attempts. University of Oulu, Degree Programme in Electronics and Communications Engineering. Master's Thesis, 72 p.

ABSTRACT

The institute of electrical and electronics engineers (IEEE) 802.11 standard specifies widely used technology for wireless local area networks (WLAN). Standard specifies high-performance physical and media access control (MAC) layers for a distributed network but lacks an effective hybrid automatic repeat request (HARQ). Currently, the standard specifies forward error correction (FEC), error detection (ED), and automatic repeat request (ARQ), but in case of decoding errors, the previously transmitted information is not used when decoding the retransmitted packet. This is called Type 1 HARQ. Type 1 HARQ uses received energy inefficiently, but the simple implementation makes it an attractive solution. Unfortunately, research applying more sophisticated HARQ schemes on top of IEEE 802.11 is limited.

In this Master's Thesis, a novel HARQ technology based on packet retransmissions that can be decoded in a turbo-like manner, keeping as much as possible compatibility with vanilla 802.11, is proposed. The proposed technology is simulated with both the IEEE 802.11 code and with the robust, efficient and smart communication in unpredictable environments (RESCUE) code. An additional interleaver is added before the convolutional encoder in the proposed technology, interleaving either the whole frame or only the payload to enable effective iterative decoding. For received frames, turbo-like iterations are done between initially transmitted packet copy and retransmissions. Results are compared against the non-iterative combining method maximizing signal-to-noise ratio (SNR), maximum ratio combining (MRC). The main design goal for this technology is to maintain compatibility with the 802.11 standard while allowing efficient HARQ. Other design goals are range extension, higher throughput, and better performance in terms of bit error rate (BER) and frame error rate (FER).

This technology can be used for range extension at low SNR range and may provide up to 4 dB gain at medium SNR range compared to MRC. At high SNR, technology can reduce the penalty from retransmission allowing higher average modulation and coding scheme (MCS). However, these gains come with the cost of computational complexity from the iterative decoding. The main limiting factors of the proposed technology are decoding errors in the header and the scrambler area, and resource-hungry-processing. In simulations, perfect synchronization and packet detection is assumed, but in reality, especially at low SNR, packet detection and synchronization would be challenging.

Keywords: Wireless, RESCUE, turbo decoding, automatic repeat request.

Paatelma A. (2021) 802.11 Pakettien Iteratiivinen dekoodaus lähetysten välillä. Oulun yliopisto, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Diplomityö, 72 s.

TIIVISTELMÄ

IEEE 802.11-standardi määrittelee yleisesti käytetyn teknologian langattomille lähiverkoille. Standardissa määritellään tehokas fyysinen- ja verkkoliityntäkerros hajautetuille verkoille, mutta siitä puuttuu tehokas yhdistetty automaattinen uudelleenlähetys. Nykyisellään standardi määrittelee virheenkorjaavan koodin, virheellisen paketin tunnistuksen sekä automaattisen uudelleenlähetyksen, mutta aikaisemmin lähetetyn paketin informaatiota ei käytetä hyväksi uudelleenlähetystilanteessa. Tämä menetelmä tunnetaan tyypin yksi yhdistettynä automaattisena uudelleenlähetyksenä. Tyypin yksi yhdistetty automaattinen uudelleenlähetys käyttää vastaanotettua signaalia tehottomasti, mutta yksinkertaisuus tekee siitä houkuttelevan vaihtoehdon. Valitettavasti edistyneempien uudelleenlähetysvaihtoehtojen tutkimusta 802.11-standardiin on rajoitetusti.

Tässä diplomityössä esitellään uusi yhdistetty uudelleenlähetysteknologia, joka pohjautuu pakettien uudelleenlähetykseen, sallien turbo-tyylisen dekoodaamisen säilyttäen mahdollisimman hyvän taaksepäin yhteensopivuutta alkuperäisen 802.11-standardin kanssa. Tämä teknologia on simuloitu käyttäen sekä 802.11- että nk. RESCUE-virheenkorjauskoodia. Teknologiassa uusi lomittaja on lisätty konvoluutio-enkoodaajan eteen, sallien tehokkaan iteratiivisen dekoodaamisen, lomittaen joko koko paketin tai ainoastaan hyötykuorman. Vastaanotetuille paketeille tehdään turbo-tyyppinen iteraatio alkuperäisen vastaanotetun kopion ja uudelleenlähetyksien välillä. Tuloksia vertaillaan ei-iteratiiviseen yhdistämismenetelmään, maksimisuhdeyhdistelyyn, joka maksimoi yhdistetyn signaali-kohinasuhteen. Tärkeimpänä suunnittelutavoitteena tässä työssä on tehokas uudelleenlähetysmenetelmä, joka ylläpitää taaksepäin yhteensopivuutta IEEE 802.11-standardin kanssa. Muita tavoitteita ovat kantaman lisäys, nopeampi yhteys ja matalampi bitti- ja pakettivirhesuhde.

Kehitettyä teknologiaa voidaan käyttää kantaman lisäykseen matalan signaali-kohinasuhteen vallitessa ja se on jopa 4 dB parempi kohtuullisella signaali-kohinasuhteella kuin maksimisuhdeyhdistely. Korkealla signaali-kohinasuhteella teknologiaa voidaan käyttää pienentämään häviötä epäonnistuneesta paketinlähetyksestä ja täten sallien korkeamman modulaatio-koodiasteen käyttämisen. Valitettavasti nämä parannukset tulevat kasvaneen laskennallisen monimutkaisuuden kustannuksella, johtuen iteratiivisesta dekoodaamisesta. Isoimmat rajoittavat tekijät teknologian käytössä ovat dekoodausvirheet otsikossa ja datamuokkaimen siemenessä. Tämän lisäksi käyttöä rajoittaa resurssisyöppö prosessointi. Simulaatioissa oletetaan täydellinen synkronisointi, mutta todellisuudessa, erityisesti matalalla signaali-kohinasuhteella, paketin tunnistus ja synkronointi voivat olla haasteellisia.

Avainsanat: Langaton, RESCUE, turbo-dekoodaus, automaattinen uudelleenlähetys

TABLE OF CONTENTS

FOREWORD

This Master's thesis has been done under the Robust, Efficient and Smart Communication in Unpredictable Environments (RESCUE) project to investigate how developed technologies would work on the commercial standard. This work has been a long runner mainly due to working full-time in parallel, but as usual, hard work pays off.

I want to thank everyone who has been helping with this work, especially Dr. Valtteri Tervo, Professor Markku Juntti, and Dr. Jiguang He. Special thanks also to everyone who worked on the RESCUE project. There were many unforgettable moments, from a demo session at the conference to drinking beer and baking sausages under the German sun. Thanks also to my girlfriend Helena, who has withstood all the weekends I have locked down myself in the office to fight against stream misalignments and corrupted packets.

Oulu, December 2nd, 2021

Anton Paatelma

LIST OF ABBREVIATIONS AND SYMBOLS

ACC-DTC	accumulator-assisted distributed turbo code
AGC	automatic gain control
ARQ	automatic repeat request
AWGN	additive white Gaussian noise
BCJR	Bahl, Cocke, Jelinek, Raviv algorithm
BER	bit error ratio
BPSK	binary phase-shift keying
CD	compact disk
CPU	central processing unit
CSMA/CA	CSMA with collision avoidance
CSMA	carrier sense multiple access
CTS	clear to send
DACC	doped accumulator
DCF	distributed coordination function
DMC	discrete memoryless channel
DSSS	direct sequence spread spectrum
ED	error detection
EXIT	extrinsic information transfer
FCS	frame check sequence
FEC	forward error correction
FER	frame error ratio
FFT	fast Fourier transform
FHSS	frequency-hopping spread spectrum
FIR	finite input response
GI	guard interval
GPU	graphics processing unit
HARQ	hybrid ARQ
HER	header error ratio
IEEE	institute of electrical and electronics engineers
IFFT	inverse FFT
I	in-phase
IIR	infinite input response
IR	incremental redundancy
ISM	industrial, scientific and medical
LDPC	low-density parity-check code
LFSR	linear feedback shift register
LLR	log-likelihood ratio
LTE	long-term evolution
MAC	media access control
MAP	maximum *a posteriori*
MCS	modulation and coding scheme
MIMO	multiple input multiple output
MN	MacKay Neal
MSDU	MAC service data unit
MU-MIMO	multi user MIMO

NAV	network allocation vector
NFC	near field communication
OFDM	orthogonal frequency division multiplexing
PDF	probability density function
PLCP	physical layer convergence procedure
PPDU	physical layer protocol data unit
PSDU	PLCP service data unit
QAM	quadrature amplitude modulation
QPSK	quadrature phase-shift keying
Q	quadrature
RESCUE	links-on-the-fly technology for robust, efficient and smart communication in unpredictable environments
RTS	ready to send
SCER	scrambler error ratio
SER	signal error ratio
SNR	signal-to-noise ratio
SOVA	soft output Viterbi algorithm
WLAN	wireless local area network

$\vec{X}$	vector $\vec{X}$, can also be lower case
X_n	Nth element of vector $\vec{X}$
$\vec{X}_n^N$	subvector of $\vec{X}$ having elements $n, n+1, \ldots, N$
$\vec{X}_{\overline{n}}$	vector $\vec{X}$ without element n
A	constant or random variable A

σ^2	variance
N_0	noise power spectral density
$\vec{u}$	vector of information bits
$\vec{v}$	vector of coded bits
$\vec{z}$	vector of samples from channel
$()_e$	extrinsic information
$()_a$	*a priori* information
$()_p$	*a posteriori* information

$\Omega_\alpha()$	Jakobi's logarithm with base α
$\Upsilon(x_1, x_2, \ldots, x_n)$	logarithmic summation of exponential terms using Jacobi's logarithm, $\ln(e^{x_1} + e^{x_2} + \cdots + x_n)$
$\Upsilon_z(x_1, x_2, \ldots)$	logarithmic summation of exponential terms for elements fulfilling condition z
$P(x)$	probability of discrete variable x or probability density function for continuous variable x
$P(x\|y)$	conditional probability of x given y
$L(x)$	log-likelihood ratio of x
$L(x\|y)$	conditional log-likelihood ratio of x given y
$p(x)$	probability density function of x
$I(x;y)$	mutual information between x and y

1. INTRODUCTION

IEEE 802.11 (later 802.11) is one of the leading wireless communication standards of today. It is commonly sold under the Wi-Fi trademark and is often used in offices, homes, and malls. It is also commonly used to provide outdoor internet services in city centers and other urban areas. 802.11 primary purpose is to provide a WLAN, although often it is used just to provide an internet access point.

The first 802.11 standard was released in June 1997, specifying only low rate (max 2Mb/s) physical layers using direct sequence spread spectrum (DSSS) and frequency-hopping spread spectrum (FHSS) technologies. The first 802.11 standard utilizing orthogonal frequency division multiplexing (OFDM) was released in September 1999. This standard is commonly known as 802.11a. This master's thesis uses an OFDM physical layer based on standard 802.11-2012. [1]

Currently, the 802.11 standard specifies FEC, ED, and ARQ, but in case of decoding errors, the previously transmitted information is not used when decoding the retransmitted packet. This thesis develops such technology capable of using retransmitted copies and the original copy of the packet in iterative decoding. As the 802.11 FEC is not optimized for this kind of use, code from RESCUE is applied on top of the 802.11 physical layer. The RESCUE code is the accumulator-assisted distributed turbo code (ACC-DTD), used initially in researches related to relayed networks allowing packets to be forwarded with bit errors [2]. The concept of forwarding erroneous bits is known as lossy forwarding. An additional interleaver is added before data scrambling in the proposed technology to enable *horizontal iterations* between packet copies.

The performance of the proposed technology is compared between FECs in terms of frame error ratio (FER) and extrinsic information transfer (EXIT). Objectives for this thesis are designing an efficient HARQ technology maintain compatibility with the 802.11 standard. Other design goals are range extension, higher throughput, and better performance in terms of bit error rate (BER) and FER.

In literature, there is plenty of other research about HARQ and iterative decoding but only limited applications on top of the 802.11 standard. One example from earlier research on area is [3], where incremental redundancy has been tested on 802.11.

This thesis is made under the RESCUE program to investigate further usages for developed coding technology. Some of the initial results with the 802.11 code were published in the RESCUE project report [4]. Rest of the thesis is organized as follows: Chapter 2 gives sufficient foundation on the theoretical side of iterative decoding, analyzing the performance of concatenated code and brief overlook on the RESCUE code. Chapter 3 briefly explains needed topics from the 802.11 standard. After building the theoretical foundation, in Chapter 4, simulator implementation is explained. Following Chapter 5 and Chapter 6, results from the EXIT analysis and simulations are gone through. Finally, results are discussed in Chapter 7 and summarized in Chapter 8.

2. ITERATIVE SOFT DECODING

In this chapter, the basics of iterative soft decoding are explained. Iterative here means that demappers and decoders are activated[1] in an iterative manner so that each decoder/demapper benefits from information coming out of other decoders/demappers. Soft here means that decoders and demappers run with "soft bits", log-likelihood ratios (LLR).

In the context of this thesis, the main interest lies in decoding convolutional codes (also used in 802.11 [1]). FECs, in general, are also briefly discussed later in this chapter to give a better understanding of the area. Probably the most classical way of decoding convolutional codes is the Viterbi decoder. Unfortunately, basic Viterbi is unusable for iterative decoding due to the hard output. For the iterative decoding, there is a soft input - soft output version of Viterbi called soft output Viterbi algorithm (SOVA) proposed by Hagenauer and Hörer in [5]. Other commonly used algorithms to perform soft decoding for convolutional codes are Bahl, Cocke, Jelinek, Raviv (BCJR) algorithm and its log-domain versions Max-Log-MAP and Log-MAP algorithms. BCJR, Max-log-MAP and Log-MAP algorithms are more closely introduced later in this chapter. In this thesis, the theoretical background of decoding convolutional codes is limited to BCJR and (Max-)Log-MAP decoders due to their better performance in terms of SNR over the SOVA. In comparison, BCJR and Log-MAP have around 0.6 dB better performance than the SOVA when used in iterative decoding, although non-iterative decoding using hard decision performance is almost identical. Better performance comes with the cost of complexity. For codes with a memory of 2, Max-log-MAP is a bit less than twice as complex as the SOVA, but it is five times more complex for codes with a long memory. [6]

Even though BCJR/Log-MAP algorithms have better performance than the SOVA, it is not optimal to decode turbo-like codes. Breiling and Hanzo proposed the optimal decoding in [7] based on the so-called Super Trellis structure. Optimal decoding is mostly ignored here because of its colossal complexity, although it has 0.35 dB better performance than iterative decoding using BCJR or Log-MAP.

In the following sections, topics related to iterative decoding are explained in detail. Firstly, an important mathematical tool, Jacobi's logarithm, is introduced in Section 2.1. Then soft demapping in Section 2.2, basics of forward error correction codes in Section 2.3. Next HARQ is explained in Section 2.4. Later decoding algorithms BCJR in Section 2.5, Max-Log-MAP in Section 2.6, and Log-MAP in Section 2.7 are explained. After necessary theoretical background, these topics are bound all together in Section 2.8, Iterative decoding. Then, in Section 2.9, the EXIT analysis, which has been shown to be an efficient tool in predicting the convergence of iterative decoding, is briefly discussed. Finally, the RESCUE code is shortly introduced in Section 2.10.

2.1. Jacobi's Logarithm

Jacobi's logarithm is well known from the theory of finite fields, also known as Zech's logarithm, named in honor of Carl Gustav Jacob Jacobi or Julius August Christoph

[1] Activation term is commonly used in coding literature, meaning running a decoder or a demapper

Zech. Its main purpose is to help sum elements over a finite field based on exponents of α. It is defined as

$$\Omega_\alpha(n) = \log_\alpha(1 + \alpha^n) \tag{1}$$

and can be used for summing exponential terms

$$\alpha^{x_1} + \alpha^{x_2} = \alpha^{x_1 + \Omega_\alpha(x_2 - x_1)}. \tag{2}$$

For this thesis following definition is used for the natural logarithm of the sum of exponential terms,

$$\Upsilon(x_1, x_2) = \ln(e^{x_1} + e^{x_2}) \tag{3a}$$
$$= x_1 + \Omega_e(x_2 - x_1) \tag{3b}$$
$$= x_2 + \Omega_e(x_1 - x_2) \tag{3c}$$
$$= \max(x_1, x_2) + \Omega_e(-|x_1 - x_2|) \tag{3d}$$

where last definition (3d) is especially useful as it minimizes exponential term inside of Ω_e, which improves numerical robustness by avoiding calculations with large numbers easily causing overflows and loss of precision. Also, this format allows easy optimization by either ignoring completely Ω_e term or replacing it with some approximation like a lookup table or a linear function. For example, in [6], it was shown that only eight values in the lookup table ranging from 0 to 5 are enough for iterative decoding, and no further improvement is achieved when using finer representation. The selected range is quite oblivious after seeing calculated Ω_e logarithm values in Figure 1. Due to the fact that the value inside of Ω_e is always negative lookup table can be relatively small.

For summations more that 2 terms Jacobi's logarithm can be used iteratively

$$\ln(e^{x_1} + e^{x_2} + \cdots + e^{x_n}) = \Upsilon(x_1, x_2, \ldots, x_n) \tag{4a}$$
$$= \Upsilon(x_1, \Upsilon(x_2, \ldots, x_n)). \tag{4b}$$

To simplify notations in forthcoming chapters following notation is used

$$\Upsilon(x_1, x_2, \ldots, x_N) \equiv \Upsilon_{n=1}^{N}(x_n). \tag{5}$$

2.2. Soft Demapping

On the transmitter side, the transmitted binary sequence is mapped before channel into a discrete constellation alphabet $f : X \to M, M \in \mathbb{C}$.[2] Therefore, on the receiver side, this operation needs to be inverted before decoding. In Section 2.2.1, the theoretical foundation of soft demapping is introduced. Later in Section 2.2.2, Gray coding and its effect for soft demapping is explained. The primary reference over the section is [8] from ten Brink et al.

[2]Symbol M is defined only for the context of this Section and redefined later for other uses.

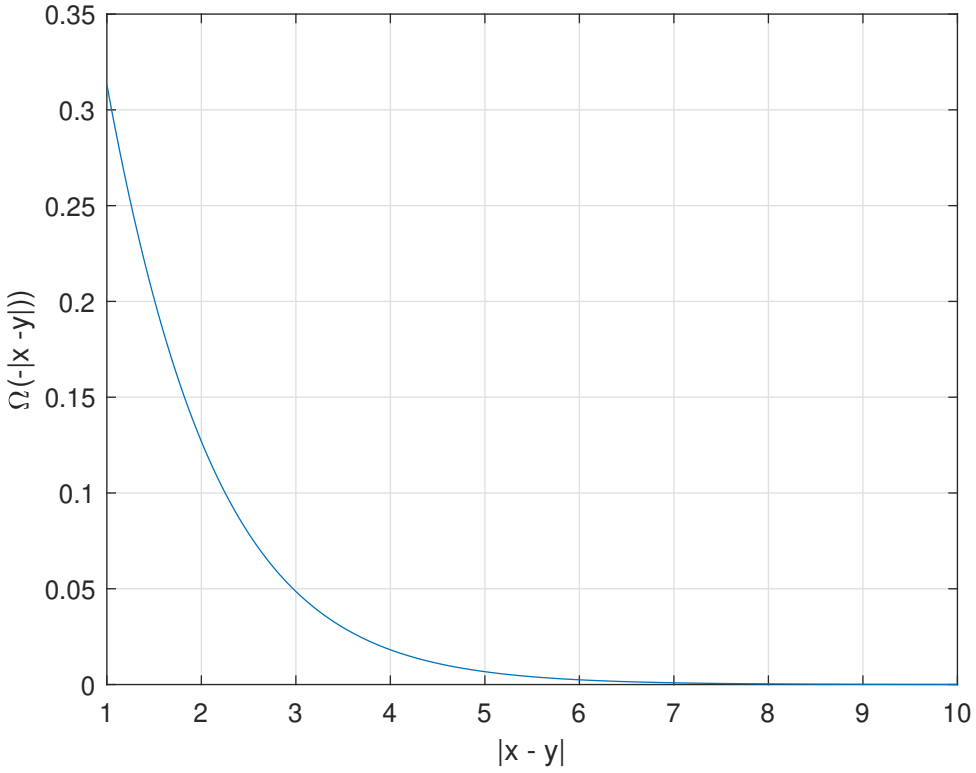

Figure 1. Jacobi logarithm as a function of $-|x-y|$

2.2.1. General Soft Demapping

In the context of this thesis, the size of the alphabet $M_1 \dots M_K$ is assumed to be 2^N, where N is an integer, which allows output an integer number of bits from the demapper per demapped symbol. In order to use all the available information, the demapper takes in *a priori* information (for example, from the decoder) and channel outputs $\vec{z}$. The demapper outputs LLRs, also known as soft bits. Because demapping is an operation run per sample, this chapter focuses on a single sample at arbitrary time instance $\vec{z}_t = z$ and demapped bits $u_1 \dots u_N$ corresponding to the demapped sample. A bit within the sample is denoted with index n. Demapper output for bit u_n, $L_e(u_n)$ is defined as,

$$L_e(u_n) = \ln(P_e(u_n)) \tag{6}$$

where $P_e(u_n)$ is defined,

$$P_e(u_n) = \frac{P(z|u_n = 1)}{P(z|u_n = 0)} \tag{7}$$

where z is a channel output.

To calculate $P_e(u_n)$, it can be noted that,

$$P_p(u_n) = \frac{P(u_n = 1|z)}{P(u_n = 0|z)} \tag{8a}$$

$$= \frac{P(u_n = 1)P(z|u_n = 1)}{P(u_n = 0)P(z|u_n = 0)} \tag{8b}$$

$$= P_a(u_n)P_e(u_n) \tag{8c}$$

where $P_p(u_n)$ is the *a posteriori* probability of a bit u_n and step (8b) follows from Bayes theorem $P(A|B)P(B) = P(B|A)P(A)$. Form in (8b) and (8c) is enough for demapping using a constellation with one bit (two possible constellation points) but for a larger constellations can be written,

$$P_e(u_n) = \frac{\sum_{\vec{A}}(P(z|u_n = 1, \vec{u}_{\overline{n}} = \vec{A})P(\vec{u}_{\overline{n}} = \vec{A})}{\sum_{\vec{A}}(P(z_t|u_n = 0, \vec{u}_{\overline{n}} = \vec{A})P(\vec{u}_{\overline{n}} = \vec{A})} \tag{9}$$

where $\vec{u}_{\overline{n}}$ is a bit vector associated with the demapped symbol without bit n. It is good to note that $P_e(u_n)$ does not depend on *a priori* information of the bit u_n.

Here the channel is assumed to be the additive white Gaussian noise (AWGN) channel. Therefore $z = x + w$, where x is the transmitted symbol and w is complex Gaussian noise with variance σ^2. Transmitted symbol x is mapped at the transmitter to a constellation point m using bit-vector $\vec{u}_n$. Noise is distributed equally between in-phase (I) and quadrature (Q) components. The I component represents the real part of the received signal whereas Q is the imaginary part. Therefore can be written

$$P(z|x) = \frac{1}{\sqrt{2\pi\sigma^2/2}} e^{-\frac{(Re(z)-Re(x))^2}{\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2/2}} e^{-\frac{(Im(z)-Im(x)^2}{\sigma^2}} \tag{10a}$$

$$= \frac{1}{\pi\sigma^2} e^{-\frac{(Re(z)-Re(x))^2+(Im(z)-Im(x))^2}{\sigma^2}} \tag{10b}$$

$$= \frac{1}{\pi\sigma^2} e^{-\frac{|z-x|^2}{\sigma^2}} = Ce^{-\frac{|z-x|^2}{\sigma^2}}. \tag{10c}$$

Now by using (10c), (9) can be rewritten. The constant term in (10c) will cancel out as it is in every term of both summations,

$$P_e(u_n) = \frac{\sum_{x,u_n=1}\left(e^{-\frac{|z-x|^2}{\sigma^2}} P(\vec{u}_{\overline{n}} = \vec{A})\right)}{\sum_{x,u_n=0}\left(e^{-\frac{|z-x|^2}{\sigma^2}} P(\vec{u}_{\overline{n}} = \vec{A})\right)} \tag{11}$$

where sum goes over all constellation points x and $\vec{A}$ is a bit-vector associated with x without demapped bit n. Now assuming that *a priori* information of $P(u_n)$ is independent from $P(u_k)$, where $k \neq n$, can be written

$$P(\vec{u}_{\overline{n}} = \vec{A}) = \prod_{k \neq n} P(u_k = a_k). \tag{12}$$

This assumption is reasonable as normally there is an interleaver between the decoder and the demapper that spreads nearby bits and breaks the dependency between them. Before applying (12) to (11) small modification can be done to (11) in order make

it more feasible for soft *a priori* information: both numerator and denominator can divided with $P(x_{\overline{n}} = \overline{0})$ and the final form is

$$P_e(u_n) = \frac{\sum_{x,u_n=1} \left(e^{-\frac{|z-x|^2}{\sigma^2}} \prod_{k \neq n, u_k=1} P_a(u_k) \right)}{\sum_{x,u_n=0} \left(e^{-\frac{|z-x|^2}{\sigma^2}} \prod_{k \neq n, u_k=1} P_a(u_k) \right)}. \tag{13}$$

Now (13) can be moved to logarithmic domain,

$$\begin{aligned} L_e(u_n) =& \ln \left(\sum_{x,u_n=1} \left(e^{-\frac{|z-x|^2}{\sigma^2} + \sum_{k \neq n, u_k=1} L_a(u_k)} \right) \right) \\ & - \ln \left(\sum_{x,u_n=0} \left(e^{-\frac{|z-x|^2}{\sigma^2} + \sum_{k \neq n, u_k=1} L_a(u_k)} \right) \right) \qquad (14a) \\ =& \Upsilon_{x,u_n=1} \left(e^{-\frac{|z-x|^2}{\sigma^2} + \sum_{k \neq n, u_k=1} L_a(u_k)} \right) \\ & - \Upsilon_{x,u_n=1} \left(e^{-\frac{|z-x|^2}{\sigma^2} + \sum_{k \neq n, u_k=1} L_a(u_k)} \right) \qquad (14b) \end{aligned}$$

where (14b) is using Jacobi's logarithm introduced earlier in Section 2.1.

2.2.2. Demapping Gray Coded Signal

Gray codes were invented by Bell Labs researcher Frank Gray in 1947 [9]. In the Gray code, two adjacent codewords differ only by one bit. In telecommunications, the signal constellation can be Gray coded, which minimizes bit errors caused by noise. The Gray coding constellation, unfortunately, provides gain only when the demapper feedback is not utilized. When the demapper loop is in use, a non-Gray constellation has much better performance.

In Grey coded quadrature amplitude modulation (QAM) constellations, I and Q are coded independently. Symbol bits are divided between I and Q, and Gray coding is applied to both of them. Therefore adjacent constellation points differ only by one bit when moving in the horizontal or vertical direction and 2 bits when moving diagonally. Because I and Q are independent, they can be also demapped independently. This reduces the complexity of demapping significantly as demapping can happen using only real numbers, and only $\sqrt{|M|}$ constellation points must be considered when calculating LLR value for a bit u_n.

2.3. Forward Error Correction Codes

Forward error correction (FEC) plays an important role not only in wireless communications but in all digital communication mediums. Forward error correction codes can either correct or correct and detect errors in the received data stream based on added redundancy. Here the main focus is on codes that can correct errors but

there are also many codes explicitly designed for error detection (ED) mainly used to implement erasure channels.

In information theory, coding can be coarsely divided into two main types, source coding, and channel coding. Source coding tries to compress data so that the source coder output bitrate is close to the entropy of input data. In many cases, like voice or video transmission, source coding can be lossy as bit-exact reproduction of input data may not be needed. In many practical lower layer applications, a scrambler is used instead of source coding as it makes data look like entropy per bit would be 1. Channel coding instead is about coding data to be suitable for transmission over a channel, which may introduce errors into data. Forward error correcting codes are practical implementations of channel coding. In the following sections, a brief history of error correcting codes and channel coding is given, and then the basics of FEC codes are presented.

2.3.1. History of Error Correction and Channel Coding

Ideas behind channel coding date back to Claude Shannon's publication in 1948 [10]. In his groundbreaking paper "A Mathematical Theory of Communication" Shannon predicted that reliable communication is asymptotically possible with an arbitrarily small error probability with channel coding. The same paper is also seen as a founding work of information theory. It introduces Shannon coding, a simple source coding technique having many similarities to the one published by Robert Fano in [11].

The first practical error correction codes were published by Golay in 1949 [12] and by Hamming in 1950 [13]. Golay proposed both binary and ternary codes in his article. Next year Hamming introduced his simple error correcting block codes. However, they were already referenced earlier by Shannon in his paper in 1948. Therefore Hamming is considered the father of error correction coding. In fact, both Shannon and Hamming worked at that time for Bell Labs. It is worth noting how early main contributions for channel coding (also source coding) were coming from a relatively small community of the Bell Labs and the Massachusetts Institute of Technology.

BCH codes were invented independently by Alexis Hocquenghem in 1959 and Raj Bose and Dwijendra K. Ray-Chaudhuri in 1960. Also in 1960, Reed-Solomon codes were published by Irving S. Reed and Gustave Solomon[14]. A well-known use case of Reed-Solomon code is the compact disk (CD) and high-speed ethernet. Special in Reed-Solomon codes is that it is not just for binary blocks. Therefore it is called non-binary block code.

All these codes are called block codes, meaning that encoding and decoding operations are applied to a block of data. In 1955 Peter Elias invented so-called convolutional codes, published in [15]. The idea in convolutional codes is simple, logical operation applied into a sliding window to generate redundancy bits. These codes are used in many practical applications, including 802.11 [1]. Convolution codes are introduced in detail later in Section 2.3.2.

In the next decade, a student of Peter Elias, Rober G. Gallager, invented low-density parity-check codes (LDPC) in 1960 in his Ph.D. dissertation. These powerful codes were long forgotten due to high computational complexity. MacKay Neal (MN) codes having many similarities with LDPC were invented in 1995 by David J.C. MacKay

and Radford M. Neal in [16]. Briefly, after publication, authors noticed similarities between MN and LDPC codes, including decoding algorithms. Like the name says, LDPC codes are based on huge but sparse parity check matrices.

Claude Berrou invented another powerful class of codes called turbo codes. The first patent for turbo codes was filed in 1991 in France [17] and later in 1993 published for academic audiences in [18]. Turbo codes are introduced in detail in Section 2.3.3.

Although many breakthroughs in this area are relatively old, polar codes were discovered in 2009 by Erdal Arikan in [19]. Polar codes rely on the design where the channel is polarized into low mutual information and high mutual information sub-channels.

2.3.2. Convolutional Codes

Convolutional codes were invented in 1955 by Peter Elias, as mentioned in Section 2.3.1. In this section basics of convolutional codes will be gone through, but for detailed description, the reader is advised to look for textbooks like [20].

The basic idea of convolutional codes is that the encoder applies modulo 2 sum operations on a sliding window of bits. Convolutional codes proposed by Elias were non-recursive. This means that there are no loopbacks in the encoder. Recursive forms of convolutional codes were presented later by Berrou et al. in their famous Turbo paper [18].

There are systematic and non-systematic convolutional codes. In systematic codes, also original bits go to the output of the encoder. The encoder may have several summation stages to generate outputs. Typically convolutional encoder is implemented with a shift register where inputs for summation are taken between shift register stages. Often these summation polynomials are expressed in an octal format where each digit contains 3 bits. Each bit represents whether the signal taken from between shift register stages is taken into summation for the specific output or feedback in recursive code. Octal notation is also used in this thesis and denoted by subscript 8. The difference between recursive and non-recursive convolutional code is similar to finite input response (FIR) filter and infinite input response (IIR) filter.

The rate of a convolutional code can be expressed as a number of output polynomials. For example, two polynomials (excluding feedback polynomials) would mean a rate of 1/2. This means that in the output of the encoder, there are two coded bits for one incoming information bit. For higher SNR ranges, usually higher code rates are wanted. Generally, this is achieved by puncturing, meaning that part of the encoder's output bits are erased before transmitting data. Puncturing patterns are designed for a specific purpose. For example, some puncturing patterns are designed to optimize decoding performance for the specific code rate. Another common design criterion is matching puncturing patterns so that difference between lower and higher rates can be expressed by additional bits added into the puncturing pattern. This kind of design is used in technology called incremental redundancy (IR).

Convolutional codes are described by parameters constraint length and free distance. Constraint length depends on the the encoder's memory i.e., the number of shift register stages in the encoder plus one. Although encoding of convolutional codes is simple using shift registers and modulo 2 summations, the decoding complexity of

optimal decoding increases exponentially with the constraint length of the code. Free distance d_{free} of the code tells about its capabilities to correct errors. It is defined as a minimum Hamming distance (number of different bits) between two encoded sequences. Although high free distance increases code error correcting capabilities, it also increases the length of error burst in case of an error in the decoding process. This is easy to understand as if the decoder makes an error in decoding, it will interpret d_{free} bits wrong. Therefore, often there is interleaver after a convolutional decoder when used as inner code in serially concatenated code to spread decoding error bursts.

Because the convolutional encoder works on a sliding window of information bits, it is essential that encoding starts and ends from/to a known state. Generally, the encoder is initialized to an all-zero state (all shift register stages are binary zero). Then, at the end of the encoding process, either zero bits are fed into the encoder in case of non-recursive code or bits that will produce zero into the shift register in case of recursive code.

Usually, possible paths through the code are visualized using a trellis. Trellis is a continuous state diagram where the horizontal axis is time, and the vertical axis is the state of the code (the content of shift register). Some convolutional codes can turn a finite number of errors in the channel into an infinite number of errors in a decoded sequence. These codes are called catastrophic.

Decoding of convolutional codes in many practical applications happens using the Viterbi algorithm, although it's not optimal. The optimal way to decode is BCJR, presented in Section 2.5, and it's corresponding log-domain versions Max-Log-MAP and Log-MAP in Section 2.6 and Section 2.7.

Doped accumulator

Doped accumulator (DACC) code is a simple code based on accumulator and doping process. Doping means that some of the coded output bits are replaced with incoming information bits. It can be seen that DACC accumulates incoming bits by running a logical XOR operation between the incoming bit and the previous outgoing bit. It selects either accumulated bit or incoming bit based on doping rate P_d. Another notable code type with accumulator is Repeat and Accumulate code [21] which is not covered here.

Due to the doping process, the DACC code rate is always one as for each input bit there is only one output bit. DACC is a special case of systematic recursive convolutional code. The accumulator can be seen as a recursive memory-1 code with loop polynomial 3_8 and output polynomial 2_8. The doping process of DACC can be seen as puncturing between systematic and accumulated outputs. Doping rate P_d of DACC is defined so that the encoder selects every P_dth bit from the accumulator output and next $P_d - 1$ systematic bits. Adding systematic bits instead of accumulated ones helps to start convergence of the decoding process. Two extremes of doping rate are $P_d = \infty$ (no accumulated bits at all) and $P_d = 1$ (only accumulated bits). The effect of doping can be easily understood by these two extremes. In the case of $P_d = \infty$ there is only a demapper which with the Gray code gives relatively high extrinsic output without any *a priori* information, but adding *a priori* does not bring much benefit. When $P_d = 1$ all the output bits are taken from the accumulator which from the extrinsic information point of view means that output information rate is low when no

a priori is given but increases sharply with *a priori* information. Doping rate $P_d = 1$ can be in some cases a bad option as accumulator alone is so-called catastrophic code, a single error bit at the beginning may swap the polarity of all the remaining bits in the decoding process. The doped accumulator is depicted in Figure 2.

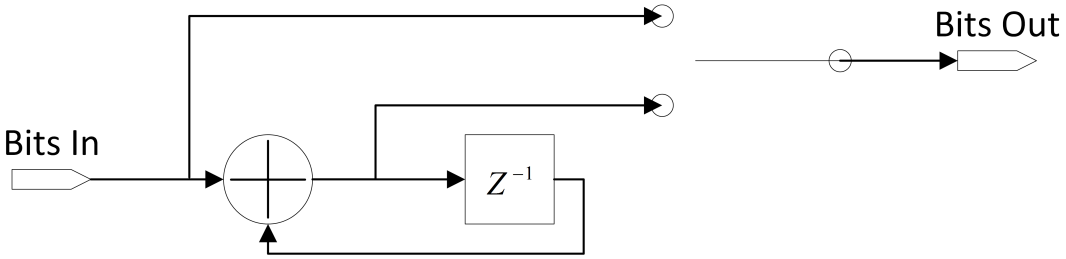

Figure 2. ACC encoder

2.3.3. Turbo Codes

Turbo codes were introduced by Berrou et al. first in a patent application in 1991 and later in 1993 in the paper as mentioned earlier in Section 2.3.1. In their landmark paper, Berrou et al. proposed an encoder structure where two systematic recursive convolutional encoders were parallel concatenated. In turbo code, encoders are separated by interleaver, and systematic bits are taken only from one encoder.

Turbo codes are decoded in an iterative manner. Iterative decoding will be discussed in detail later in Section 2.8. Common decoding algorithms for turbo codes are Max-Log-Map and SOVA. Decoding is done separately for each component code, and results are used as *a-priori* information for another decoder.

Turbo codes are called capacity-achieving as they have performance close to the Shannon limit. Turbo codes have characteristic BER behavior, steep "Turbo cliff" where BER falls fast as a function of SNR. After this, there is the so-called BER floor, which can be predicted from the code's free distance [22].

2.4. Hybrid Automatic Repeat Request

Hybrid automatic repeat request (HARQ) is the more advanced version of ARQ. Although it has been proven that having feedback in the communication channel cannot increase capacity, it can significantly simplify the encoding and decoding of transmitted data [23]. Simplification is easy to understand: If there is feedback available, the transmitter can just send erroneous bits again instead of extremely long code blocks to correct errors caused by the channel.

In classical ARQ technology, a transmitted packet carries ED code (often cyclic redundancy check code) to detect erroneous packets. By using a feedback channel, the receiver informs the transmitter about erroneous packets which are resent. In HARQ, there is also a FEC code to correct possible errors caused by channel [24].

In literature, HARQ is commonly divided into three subtypes:

1. Sending the same data content on every retransmission.
2. Sending more redundancy on each re-transmissions.
3. Sending more redundancy on each re-transmission, but each re-transmission is self-decodable.

In modern type 2 and 3 HARQ technologies, a smaller amount of information is transmitted on the first attempt, and if decoding fails, more redundancy is sent. After the transmission, the decoder tries to decode the received packet using both original and new information. Notable type 2 HARQ technology is Incremental Redundancy (IR).

2.5. Bahl, Cocke, Jelinek, and Raviv Decoder

Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm was originally presented in 1974 by Bahl, Cocke, Jelinek, and Raviv in their paper [25]. The algorithm is named after its inventors. It is a maximum *a posteriori* (MAP) algorithm that estimates the *a posteriori* probabilities (APP) of a Markov source observed through a discrete memoryless channel (DMC). BCJR algorithm can be used to decode linear block codes and convolutional codes, and it minimizes symbol (or typically bit) error probability.

Because the MAP algorithm is complex and its performance in decoding convolutional codes with hard decision is comparable to the Viterbi decoder, it was long forgotten until the introduction of turbo codes [26]. In this section, derivation of the algorithm in its original form is followed, except some parts are modified to be more suitable for iterative decoding where input may come from a soft demapper or another decoder. In the following sections, it is extended to its modern formats operating in the logarithmic domain.

Symbol source is discrete-time finite-state Markov process having M states. State at time t is denoted by S_t and output is v_t, where v_t belongs to a finite discrete alphabet. Source starts from state $S_0 = 0$, produces an output sequence $\vec{v}_1^{\tau} = \vec{v}$, ending into the state $S_t = 0$. The sequence $\vec{v}$ goes through a DMC[3] and produces output $P_a(\vec{v})$, which is used as an input of the decoder. The decoder tries to estimate APP of states and transitions, i.e., conditional probabilities

$$P(S_t = m|\vec{v}) \tag{15}$$

and

$$P(S_{t-1} = m'; S_t = m|\vec{v}). \tag{16}$$

By using Bayes rule (15) and (16) can re-written

$$P(S_t = m, \vec{v})/P_a(\vec{v}) \tag{17}$$

and

$$P(S_{t-1} = m'; S_t = m, \vec{v})/P_a(\vec{v}). \tag{18}$$

[3]In this context, the channel may not be only the actual physical channel, but may also include decoders and a demapper.

Because $P_a(\vec{v})$ is a constant it can be ignored as normal output of BCJR algorithm is likelihood ratios where $P(\vec{v})$ will cancel out because it is in both numerator and denominator. If actual transition probabilities or state probabilities are needed, sum can be normalized to 1. Lets define helper variable for probability of state m at time t,

$$\lambda_t(m) = P(S_t = m, \vec{v}) \tag{19a}$$
$$= P(S_t = m, \vec{v}_1^t) P(\vec{v}_{t+1}^\tau | S_t = m, \vec{v}_1^t) \tag{19b}$$
$$= P(S_t = m, \vec{v}_1^t) P(\vec{v}_{t+1}^\tau | S_t = m) \tag{19c}$$

where (19c) follows from the fact that for Markov source $P(S_t|S_1^{t-1}) = P(S_t|S_{t-1})$ and because $\vec{v}_1^t$ are related to state transitions before t. A helper variable for state transitions between time $t-1$ and t is defined:

$$\sigma_t(m', m) = P(S_{t-1} = m', S_t = m, \vec{v}) \tag{20a}$$
$$= P(S_{t-1} = m'; \vec{v}_1^{t-1}) P(\vec{v}_t^\tau, S_t = m | S_{t-1} = m', \vec{v}_1^{t-1}) \tag{20b}$$
$$= P(S_{t-1} = m'; \vec{v}_1^{t-1}) P(\vec{v}_t^\tau, S_t = m | S_{t-1} = m') \tag{20c}$$
$$= P(S_{t-1} = m', \vec{v}_1^{t-1}) P(\vec{v}_t, S_t = m | S_{t-1} = m') P(\vec{v}_{t+1}^\tau | S_t = m). \tag{20d}$$

The last steps (20c) and (20d) follow similarly from the properties of a Markov source.

Now lets define helper variables α, β and γ

$$\alpha_t(m) = P(S_t = m, \vec{v}_1^t) \tag{21}$$

$$\beta_t(m) = P(\vec{v}_{t+1}^\tau | S_t = m) \tag{22}$$

$$\gamma_t(m', m) = P(S_t = m, v_t | S_{t-1} = m'). \tag{23}$$

Now using α, β and γ, equations (19c) and (20d) can be rewritten:

$$\lambda_t(m) = \alpha_t(m) \beta_t(m) \tag{24}$$

$$\sigma_t(m', m) = \alpha_{t-1}(m') \gamma_t(m', m) \beta_t(m). \tag{25}$$

Next, α, β and γ need to be opened up to a more usable iterative format. For α, the iterative form is

$$\alpha_t(m) = \sum_{m'} P(S_{t-1} = m', S_t = m, \vec{v}_1^t) \tag{26a}$$
$$= \sum_{m'} P(S_{t-1} = m', \vec{v}_1^{t-1}) P(S_t = m, v_t | S_{t-1} = m') \tag{26b}$$
$$= \sum_{m'} \alpha_{t-1}(m') \gamma_t(m', m). \tag{26c}$$

For initialization of iteration $\alpha_0(0) = 1$ and $\alpha_0(m) = 0, m \neq 0$. This follows from the definition $S_0 = 0$. The iterative form of β is

$$\beta_t(m) = \sum_{m'} P(S_{t-1} = m', \vec{v}_{t+1}^{\tau} | S_t = m) \tag{27a}$$

$$= \sum_{m'} P(S_{t-1} = m', v_{t+1} | S_t = m) P(\vec{v}_{t+2}^{\tau} | S_{t+1} = m') \tag{27b}$$

$$= \sum_{m'} \beta_{t+1}(m') \gamma_{t+1}(m, m'). \tag{27c}$$

Similarly initialization $\beta_\tau(0) = 1$ and $\beta_\tau(m) = 0, m \neq 0$. Finally the iterative form of γ is

$$\gamma_t(m', m) = P(S_t = m, v_t | S_{t-1} = m') \tag{28a}$$

$$= P(v_t | S_t = m, S_{t-1} = m') P(S_t = m, S_{t-1} = m') \tag{28b}$$

$$= P_a(v_t = x) P(S_t = m, S_{t-1} = m'). \tag{28c}$$

In equation (28c), the term $P(S_t = m | S_{t-1} = m')$ is *a priori* probability for a transition from state m' to state m. It is the same as the information bit probability $P_a(u_t)$ associated with the transition and becomes zero for impossible transitions. This is an extremely important property in trellis-aided decoding as it allows us to ignore all the transitions not specified by the trellis. The last step follows from the fact that $S_t = m, S_{t-1} = m'$ defines also transmitted channel bits (denoted here with x), and the same conditioning happens already in the demapper.

When using LLRs as input $P(u_t)$ can be expressed,

$$P_a(u_t = \pm 1) = \frac{e^{\pm L_a(u_t)/2}}{e^{L_a(u_t)/2} + e^{-L_a(u_t)/2}} \tag{29a}$$

$$= C_{ut} e^{\pm L_a(u_t)/2} \tag{29b}$$

where $L_a(u_t)$ is *a priori* LLR of information bit at time t. Because all $\gamma_t(m', m)$ terms have the same common multiplier C_{ut} it will cancel out in later calculations. Therefore $P_a(u_t = \pm 1) = e^{\pm L_a(u_t)/2}$ can be used instead. $L_a(u_t)$ may come from demapper in case of systematic bit or from another decoder as *a priori* information. This topic will be handled in Section 2.8.

Because in this context transmission is binary, each v_t can be further divided into N bits noted by additional subscript. Similarly $P_a(v_t)$ can be calculated as

$$P_a(v_t = x) = \prod_{n=1}^{N} P_a(v_{tn} = x_n) \tag{30a}$$

$$= \prod_{n=1}^{N} \frac{e^{x_n L_a(v_{tn})/2}}{e^{L_a(v_{tn})/2} + e^{-L_a(v_{tn})/2}} \tag{30b}$$

$$= \prod_{n=1}^{N} C_{vtn} e^{x_n L_a(v_{tn})/2} \tag{30c}$$

where N is number of coded bits per information bit and $L_a(v_{tn})$ is *a priori* LLR associated for coded bit n at time t. Here there is also a common term C_{vtn} not dependent on the polarity of the bit x_n and therefore constants C_{vtn} can be ignored from

the calculation as they will cancel out later. $L_a(v_{tn})$ may come from soft demapper (already discussed in Section 2.2) or from another decoder in the case of serially concatenated turbo codes. Using this information final form for $\gamma_t(m', m)$ can be written

$$\gamma_t(m', m) = e^{u_t L_a(u_t)/2 + \sum_v^N (v_{tn} L_a(v_{tn})/2)}. \tag{31}$$

When BCJR is used to decode convolutional codes we are interested to get log likelihood ratios of transmitted bits $\ln(P_p(u_t = 1)/P_p(u_t = 0))$. Calculation for *a posteriori* LLRs is

$$L_p(u_t) = \ln\left(\frac{\sum_{u_t=1} \sigma_t(m', m)}{\sum_{u_t=-1} \sigma_t(m', m)}\right). \tag{32}$$

The nominator part has all possible transitions $m' \rightarrow m$, where information bit $u_t = 1$ is associated and denominator all transitions with $u_t = -1$. *A posteriori* LLR:s for coded bits $L_p(v_{tn})$ can be calculated similarly by selecting transitions where coded bit v_{tn} is associated, nominator composed from transitions where $v_{tn} = 1$, and denominator from transitions where $v_{tn} = -1$.

2.6. Max-Log-MAP Decoder

Because of the high complexity of the MAP algorithm, there are many simplifications proposed over time. The most known of them are Max-Log-MAP and Log-MAP algorithms. In this section, the idea behind the Max-Log-MAP decoder is briefly gone through, and as natural continuity, in Section 2.7 Log-MAP decoder is introduced.

Max-Log-MAP was introduced in several different references between 1989-1990 [26]. The basic idea behind Max-log-MAP is to move calculations of the BCJR algorithm into the logarithmic domain and exploit the fact that $\ln(e^x + e^y) \approx \max(x, y)$. Unfortunately, this approximation is not valid when x and y are close to each other, which causes some degradation in performance.

First equations (26c), (27c) and (28c) can be redefined in logarithmic domain

$$A_t(m) = \ln(\alpha_t(m)) \tag{33}$$

$$B_t(m) = \ln(\beta_t(m)) \tag{34}$$

$$\Gamma_t(m, m') = \ln(\gamma_t(m, m')). \tag{35}$$

The iterative form of A becomes

$$A_t(m) = \ln\left(\sum_{m'} (\alpha_{t-1}(m')\gamma_t(m', m))\right) \tag{36a}$$

$$= \ln\left(\sum_{m'} e^{A_{t-1}(m') + \Gamma_t(m', m)}\right) \tag{36b}$$

$$\approx \max_{m'}(A_{t-1}(m') + \Gamma_t(m', m)). \tag{36c}$$

Similarly for $B_t(m)$

$$B_t(m) = \ln\left(\sum_{m'}(\beta_{t+1}(m')\gamma_{t+1}(m,m'))\right) \tag{37a}$$

$$= \ln\left(\sum_{m'} e^{B_{t+1}(m')+\Gamma_{t+1}(m,m')}\right) \tag{37b}$$

$$\approx \max_{m'}(B_{t+1}(m') + \Gamma_{t+1}(m,m')). \tag{37c}$$

And for $\Gamma_t(m,m')$

$$\Gamma_t(m,m') = \ln\left(e^{u_t L_a(u_t)/2+\sum_v^N (x_n L_a(v_{tn})/2)}\right) \tag{38a}$$

$$= u_t L_a(u_t)/2 + \sum_v^N (x_n L_a(v_{tn})/2). \tag{38b}$$

In Section 2.5 $\sigma_t(m',m)$ is used to calculate output LLRs. It can be calculated using its logarithmic counterparts

$$\sigma_t(m',m) = \alpha_{t-1}(m')\gamma_t(m',m)\beta_t(m) \tag{39a}$$

$$= e^{\ln(\alpha_{t-1}(m')\gamma_t(m',m)\beta_t(m))} \tag{39b}$$

$$= e^{A_{t-1}(m')+\Gamma_t(m',m)+B_t(m)}. \tag{39c}$$

Finally, by using approximation $\ln(e^x + e^y) \approx \max(x,y)$, LLR calculation can be written

$$L_p(u_t) = \ln\left(\frac{\sum_{u_t=1}\sigma_t(m',m)}{\sum_{u_t=-1}\sigma_t(m',m)}\right) \tag{40a}$$

$$= \ln\left(\frac{\sum_{u_t=1} e^{A_{t-1}(m')+\Gamma_t(m',m)+B_t(m)}}{\sum_{u_t=-1} e^{A_{t-1}(m')+\Gamma_t(m',m)+B_t(m)}}\right) \tag{40b}$$

$$\approx \max_{u_k=1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)) - \max_{u_k=-1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)). \tag{40c}$$

$L_p(v_{tn})$ can be then calculated similarly

$$L_p(v_{tn}) \approx \max_{v_n=1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)) - \max_{v_n=-1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)). \tag{41}$$

2.7. Log-MAP Decoder

After taking the approximation $\ln(e^x + e^y) \approx \max(x,y)$, the natural question is can we do better? This question was answered by Robertson *et al.* in [6]. The proposed improvement was to replace $\ln(e^x + e^y) \approx \max(x,y)$ with

$$\ln(e^x + e^y) = \max(x,y) + \ln(1 + e^{-|x-y|}) \tag{42a}$$

$$= \max(x,y) + \Omega_e(-|x-y|) \tag{42b}$$

$$\approx \max(x,y) + f_c(|x-y|) \tag{42c}$$

where Ω_e is Jacobi's logarithm (introduced in Section 2.1) and f_c is approximation of it. An ideal Log-MAP decoder would use Jacobi's logarithm as it but normally it is replaced by computationally lightweight approximation.

Now equations (26c), (27c) can be rewritten to calculate $A_t(m)$, $B_t(m)$ for Log-MAP decoder

$$A_t(m) = \ln\left(\sum_{m'} e^{A_{t-1}(m')+\Gamma_t(m',m)}\right) \tag{43a}$$
$$= \Upsilon_{m'}(A_{t-1}(m') + \Gamma_t(m',m)) \tag{43b}$$

$$B_t(m) = \ln\left(\sum_{m'} e^{B_{t+1}(m')+\Gamma_{t+1}(m,m')}\right) \tag{44a}$$
$$= \Upsilon_{m'}(B_{t+1}(m') + \Gamma_{t+1}(m,m')). \tag{44b}$$

Finally for *a posteriori* LLR:s

$$L_p(u_t) = \ln\left(\frac{\sum_{uk=1}\sigma_t(m',m)}{\sum_{uk=-1}\sigma_t(m',m)}\right) \tag{45a}$$
$$= \ln\left(\frac{\sum_{u_t=1} e^{A_{t-1}(m')+\Gamma_t(m',m)+B_t(m)}}{\sum_{u_t=-1} e^{A_{t-1}(m')+\Gamma_t(m',m)+B_t(m)}}\right) \tag{45b}$$
$$= \Upsilon_{u_t=1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)) - \Upsilon_{u_t=-1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)). \tag{45c}$$

$L_p(v_{tn})$ can be then calculated similarly

$$L_p(v_{tn}) = \Upsilon_{v_n=1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)) \tag{46}$$
$$- \Upsilon_{v_n=-1}(A_{t-1}(m') + \Gamma_t(m',m) + B_t(m)). \tag{47}$$

2.8. Iterative Decoding

Iterative decoding plays a big role in modern communication systems utilizing turbo codes. The basic idea in iterative decoding is to activate decoders and demappers iteratively. Each decoder taking in extrinsic information from other decoders as *a priori* information and/or intrinsic information from the demapper. By using *a priori* information, each decoder is able to output more extrinsic information, which is again used by next decoder. By running multiple iterations, decoders' output converges towards error free decoding.

In turbo coding literature, turbo codes are divided into two main categories, parallel concatenated codes and serially concatenated codes. Example encoders can be seen in Figure 3 and corresponding decoders in Figure 4. In Figure 4 $L_e(\vec{v})$ is extrinsic information of coded bits, $L_a(\vec{v})$ is *a priori* information of coded bits. Similarly, $L_e(\vec{u})$ and $L_a(\vec{u})$ are extrinsic and *a priori* information of information bits. $L_p(\vec{u})$ is *a posteriori* information of the whole decoder.

In parallel concatenated codes, incoming bits are duplicated into two streams. One stream goes directly to an encoder, and another first to an interleaver and then to an encoder. Normally these encoders are the same, but they can also be different. Usually,

there is puncturing after encoders, dropping some of the encoded bits to increase the code rate. In serially concatenated code, incoming data goes first to one encoder, then after interleaver, to another encoder. Commonly in serially concatenated codes, encoders are different (known as inner and outer encoders). Usually, turbo codes use convolutional codes as component codes.

An important thing to note is that all the codes are always separated by interleaver to make data coming out of decoders as independent as possible from nearby bits before entering into the next decoder. This means that bits close to each other in the input of an interleaver are spread over a longer period in output. In practical implementations, interleavers are often block interleavers, where data is written to rows and read from columns. Another commonly used practical interleaver type is the convolutional interleaver which divides bit sequence into different length queues [20]. In academic research also random interleavers where permutation is a random function are popular. In the example encoder in Figure 4, there are interleavers only between the codes, but in practice, it can be beneficial to put an interleaver between the demapper and first decoder. This is because the demapper can also be understood as a kind of decoder having memory between bits and being part of the iterative process. Interleaver between demapper and decoder may also protect from burst errors coming from the channel. More about interleavers can be read from [27] and from [28].

All the component decoders and demappers are passing to other decoders/demappers only extrinsic information. In general, one can say that $L_p = L_a + L_e$, where L_p is *a posteriori* information in the logarithmic domain and L_a and L_e respectively *a priori* information and extrinsic information. This means that each of the decoders/demappers either removes *a priori* information from its output or calculates directly L_e that is independent of *a priori* information L_a to avoid a loop, where output information of the decoder comes back as *a priori* information through another decoder. The final decision, whether a bit is 1 or 0, is based on *a posteriori* information. Also notable in Figure 4 is that output $L_p(\vec{u})$ can be the sum of decoders extrinsic information or directly $L_p(\vec{u})$ of some component decoder. In both cases, the result is the same as long as $L_p(\vec{u})$ comes from the last activated decoder.

Usually, multiple iterations are needed. As decoding complexity increases linearly with the number of iterations, it is important to run only as many iterations as needed. Some sources, like [26], state that a maximum of 8 iterations are needed, and no significant improvement is seen after that. Decoding complexity can be reduced greatly by using some end condition for iteration together with a maximum iteration limit. Suitable criteria to terminate the decoding process can be cross-entropy of outputs of component decoders like proposed in [29] or variance of a component decoder $L_p(\vec{u})$ output as proposed in [30]. Component decoder output variance can also be used to estimate mutual information. This topic is covered in Section 2.9.

One of the most significant issues in iterative decoding is the long processing time. Interleavers and decoders may require that all the data is available before processing, and multiple iterations may require lots of time. A common way to decrease processing time is to use limited code block size and limit iterations. For example, long-term evolution (LTE) has a limited code block size of 6144 bits [31]. Unfortunately, smaller block size leads to smaller interleavers which has been proven to decrease performance [22].

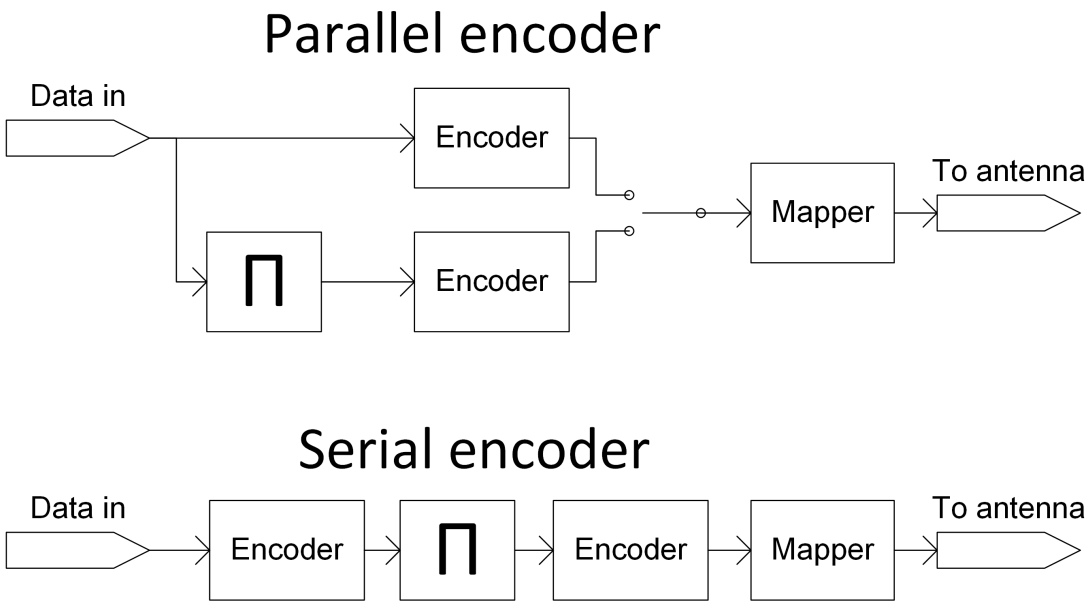

Figure 3. Example serial and parallel concatenated turbo encoders

2.9. Extrinsic Information Transfer Analysis

This section explains the essential theoretical background behind the extrinsic information transfer (EXIT) analysis. The EXIT analysis is a tool that can be used to predict the performance and the convergence of iterative decoding. It was initially introduced by Stephan ten Brink in [32] and further discussed in [33].

Like the name says, the EXIT analysis is based on extrinsic information transfer between decoders, demappers, equalizers, and other processing blocks having memory. Usually, output from the EXIT analysis is presented in a EXIT chart, where one axis of the drawn chart is incoming *a priori* information, and another is outgoing extrinsic information. EXIT analysis can be seen as an evolution for early SNR-based methods like the one in [34]. The EXIT analysis shows how *a priori* information at soft decoder's input converts into extrinsic information at soft decoder's output.

In Section 2.9.1 and Section 2.9.2 theoretical foundation of the EXIT analysis will be briefly gone through. Then in Section 2.9.3, simulation of an EXIT chart will be presented.

2.9.1. Theory of Extrinsic Information Transfer Analysis

Exit analysis measures mutual information between extrinsic information output of the decoder and correct bit sequence. Often results are presented as a function of *a priori*, giving more insight of performance in iterative decoding.

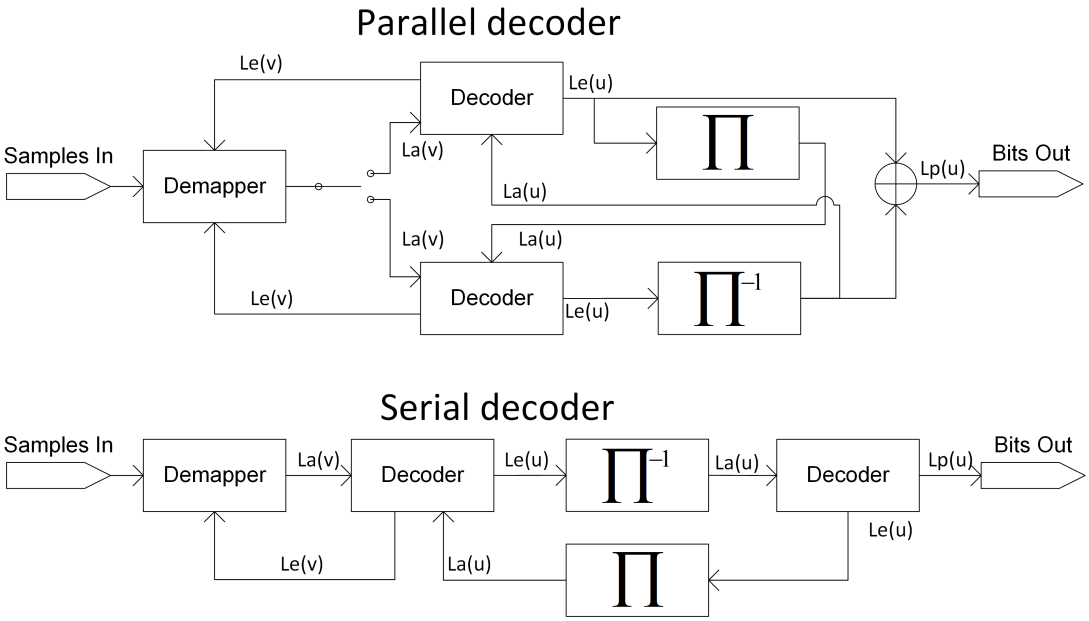

Figure 4. Example serial and parallel concatenated turbo decoders

LLR output of a demapper and a decoder after several iterations is Gaussian [35 pp. 49], as well as soft demapper output as seen in Section 2.2. Soft demapper for binary phase-shift keying (BPSK) outputs LLRs with mean

$$\mu_z = 2/\sigma_n^2 \tag{48}$$

where σ_n^2 is a variance of demapper output. The Standard deviation of LLRs is

$$\sigma_z = \sqrt{4/\sigma_n^2}, \tag{49}$$

meaning that variance and mean of LLRs at demapper output are bounded by equation

$$\mu_z = \frac{\sigma_z^2}{2}. \tag{50}$$

This property is critical when generating *a priori* information for EXIT analysis.

Mutual information between binary random variable X, and continuous random variable Y is defined

$$I(X;Y) = \sum_{x=+1,-1} \int_{-\infty}^{\infty} P(y|x)P(x) \log \frac{P(y,x)}{P(y)} dy \tag{51}$$

where $P(y|x)$ is the probability density function (PDF) of y given x and $P(x)$ is the probability of x. For equally likely X, $P(x) = 1/2$ can be taken outside which simplifies equation (51) into form

$$I(X;Y) = \frac{1}{2} \sum_{x=+1,-1} \int_{-\infty}^{\infty} P(y|x) \log \frac{P(y,x)}{P(y)} dy. \tag{52}$$

For LLRs PDF is symmetric, $P(y|x=1) = P(-y|x=-1)$. For equally likely x can be written $P(y)$

$$P(y) = \frac{1}{2}(P(y|x=1) + P(y|x=-1)) \tag{53}$$

and

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-x)^2}{2\sigma^2}}. \tag{54}$$

LLR with symmetric distribution satisfies the consistency condition

$$P(-y|x) = e^{-L_c xy} P(y|x) \tag{55}$$

where $L_c xy$ is the channel reliability value.

These conditions simplify the estimation of mutual information. For equally likely X and LLR:s with symmetric and consistent distribution, mutual information can be rewritten

$$I(X; L_e(X)) = 1 - \int_{-\infty}^{\infty} P(L|x=1) \log_2(1 + e^{-xL_e(x)}) \tag{56a}$$

$$= 1 - E(\log_2(1 + e^{-xL_e(x)})) \tag{56b}$$

where $E()$ is the expectation. Because of law of large numbers expectation can be estimated by sample average

$$I(X; L_e(X)) \approx 1 - \frac{1}{N} \sum_{t=1}^{N} \log_2(1 + e^{-x_t L_e(x_t)})). \tag{57}$$

It is also possible to estimate information without knowing sequence $\vec{x}$ by observing that $x_t \approx sgn(L_e(x_t)$. Decoding error occurs with probability

$$P(sgn(L_e(x_t)) \neq x) = \frac{e^{-|L_e(x_t)|/2}}{e^{|L_e(x_t)|/2} + e^{-|L_e(x_t)|/2}} \tag{58a}$$

$$= \frac{\tanh(-|L_e(x_t)|/2) + 1}{2} \tag{58b}$$

where the absolute value of $Le(x_t)$ is not needed in the denominator but is added because it does not change the result and allows using the form in (58b). Therefore (57) can be rewritten into form

$$I(X; L_e(X)) \approx 1 - \frac{1}{N} \sum_{t=1}^{N} H_b(\frac{\tanh(-|L_e(x_t)|/2) + 1}{2}) \tag{59}$$

where H_b is binary entropy

$$H_b(p) = -p \log_2(p) - (1-p) \log_2(1-p). \tag{60}$$

2.9.2. J-Function

A priori information can be generated with the help of the so-called J-function, introduced in [33]. It can be used to calculate the variance of *a priori* signal. J-function is defined

$$J(\sigma) \equiv I_A(\sigma_A = \sigma) \tag{61}$$

and

$$\sigma_A = J^{-1}(I_A)). \tag{62}$$

J-function is the same as the equation (56a), expressed as a function of σ. Unfortunately, J-function is not possible to be calculated in closed form, but it can be closely approximated by [36]

$$J(\sigma) \approx (1 - 2^{-H_1\sigma^{2H_2}})^{H_3} \tag{63}$$

and

$$J^{-1}(I_A) \approx \left(-\frac{1}{H_1} log_2\left(1 - I_A^{\frac{1}{H_3}}\right)\right)^{\frac{1}{2H_2}}. \tag{64}$$

Numerically optimized coefficients to minimize squared error for equation (63) are provided by [37]: $H_1 = 0.3073$, $H_2 = 0.8935$ and $H_3 = 1.1064$. Calculated J-function can be seen in Figure 5.

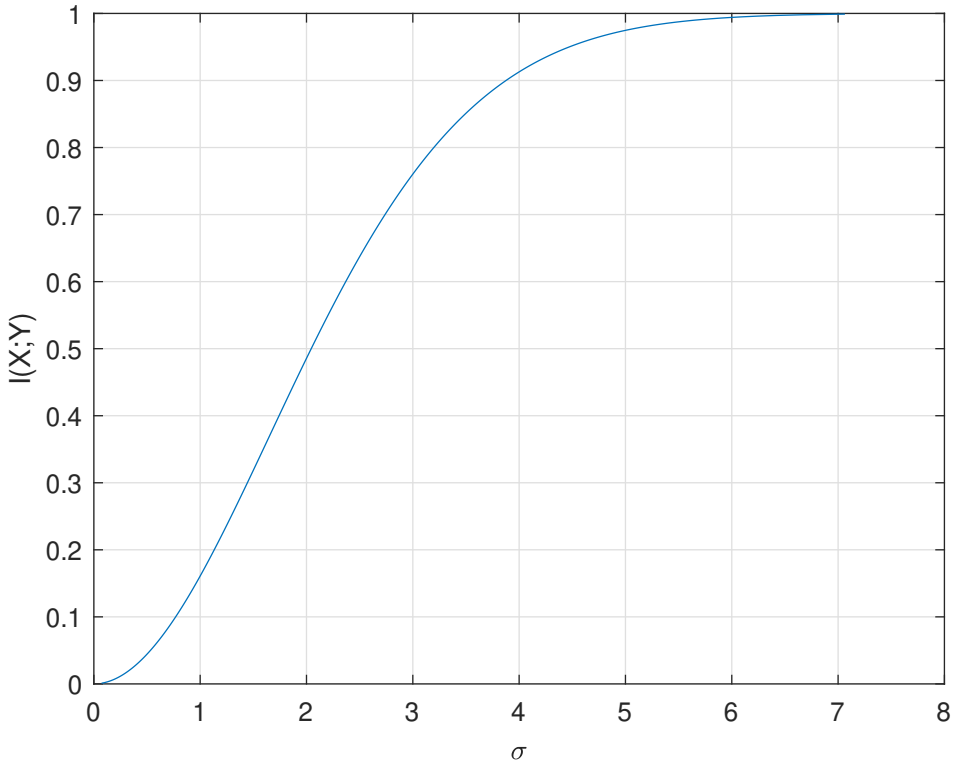

Figure 5. J-function approximation calculated using equation 63

2.9.3. Extrinsic Information Transfer Chart Simulation

For simulation, *a priori* information can be generated by feeding generated bit sequence $\vec{x} \in \{-1, 1\}$ through an AWGN channel. Required signal variance can be calculated using inverse J-function in equation (64). Variance can be split into mean

and standard deviation using formulas (48) and (49). The calculated mean is AWGN channel gain, and the standard deviation is AWGN channel standard deviation. After decoder, LLRs are multiplied by noiseless *a priori* information (original sequence) and then fed to mutual information estimation using formula (57). This procedure is depicted in Figure 6. Another option is to ignore multiplication by original sequence and trust only LLRs outputted by decoder using formula (59). This approach relies on correctly calculated LLRs and can lead to false results if the decoder outputs higher LLR values than actual reliability is.

Normally the EXIT chart used to investigate concatenated code convergence is drawn so that the EXIT curve of the second decoder (or demapper etc.) is drawn axes swapped, *a priori* information on another axis that first decoder's and outcoming extrinsic information respectively. Sometimes the EXIT chart also includes a decoding trajectory, averaged information transfer from the actual decoding process.

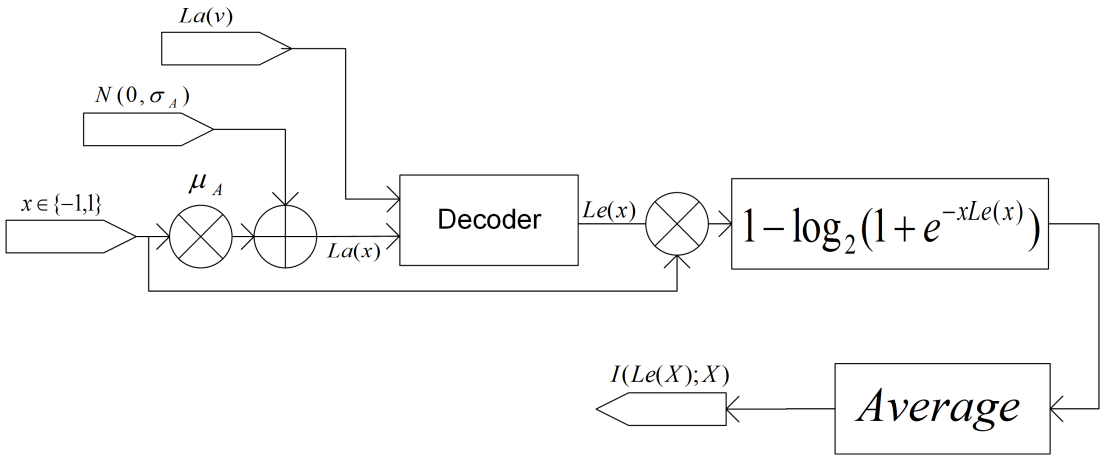

Figure 6. Basic concept of EXIT analysis

2.10. Rescue Code

The RESCUE code and physical layer were developed for links-on-the-fly technology for robust, efficient and smart communication in unpredictable environments (RESCUE) project which were mainly studying possible gains of lossy forwarding in unpredictable environments like catastrophe zones where infrastructure is more or less destroyed. Code and physical layer were used in simulations, conference demo and over the air (OTA) verification using USRP software radios. [38] [4]

The RESCUE code is designed for joint decoding between several copies of the same information coming from multiple relay nodes and/or from the original transmitter, forwarded either lossy (not all bits decoded successfully) or lossless. Code is known as accumulator-assisted distributed turbo code (ACC-DTC) [2], where distributed comes from the fact that the original use-case for the code was to be used in relayed systems.

There are two convolutional component codes in the RESCUE code: The outer code is a non-recursive non-systematic memory-2 convolutional code with octal polynomials 7_8 and 5_8. The inner code is DACC. Between these two encoders, there

is an interleaver to randomize bit order. For the DACC, RESCUE uses two different doping rates, P_d, depending on MCS: 8 for quadrature phase-shift keying (QPSK) and 1 for 16-QAM. Rescue coding and decoding is depicted in Figure 7.

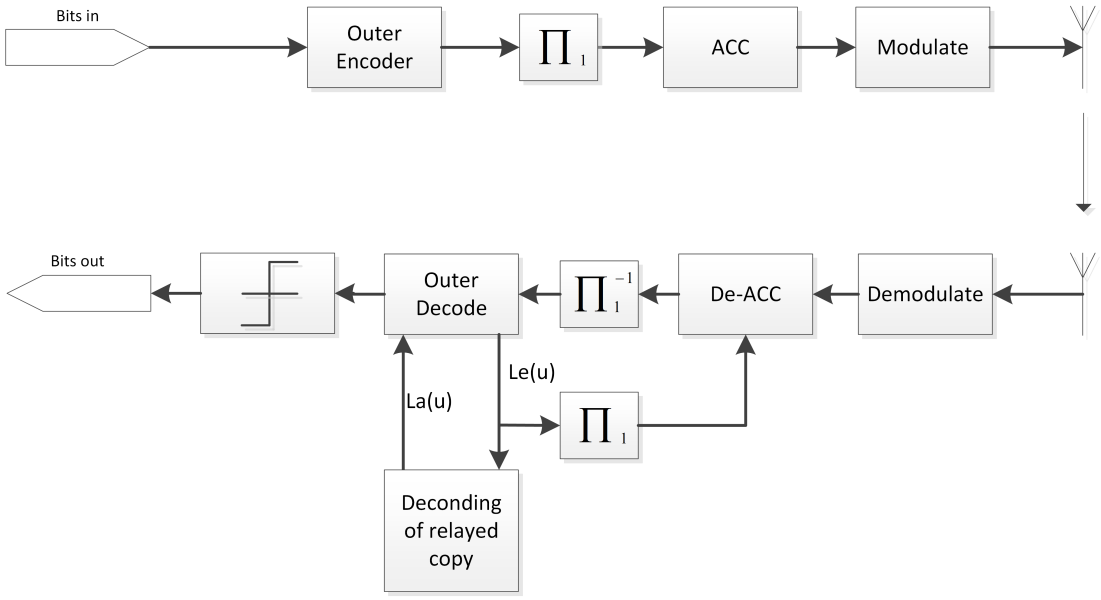

Figure 7. Rescue encoder and decoder

3. IEEE 802.11 PHYSICAL LAYER AND MEDIA ACCESS CONTROL

In this chapter, the basics of the 802.11 (trademark Wi-Fi) physical layer and some parts of MAC relevant to this thesis are explained. 802.11 is the most common wireless local area network (WLAN) technology used both in homes and public areas. Nowadays most companies offer employees wireless access to the company's network using this technology as well as many restaurants are using it to attract customers. Probably everyone has seen "Free Wifi" signs.

In some densely populated areas, cellular operators are also using 802.11 to offload traffic from the cellular network. This is an easy option for an operator as 802.11 operates on non-licensed industrial, scientific and medical (ISM) bands free to use.

As ISM bands are mainly free to use, communication on bands reserved for mobile communication is mainly interference-limited (except in rural areas). Centimeter wave Bands used by 802.11 are located around 2.4 GHz and 5.8 GHz frequencies. On very same bands, there are operating multiple different technologies like Bluetooth, 802.15.4 (traded as Zigbee), and near field communication (NFC). Communication of this kind of crowded band places many requirements for MAC- and Physical -layer and is also an active area for scientific research. Machine learning may also give great benefits for stationary / slowly changing environments like indoor use [39].

Current standards of 802.11 include many state-of-art technologies like up to 8 layers multiple input multiple output (MIMO), multi user MIMO (MU-MIMO), bandwidth up to 160Mhz. However, instead of all these state-of-the-art technologies, only simplified single input single output (SISO) operation of the older 802.11n standard is covered due to relevance for this thesis.

This chapter is organized as follows: Firstly, Section 3.1 the 802.11 OFDM physical layer will be discussed. Later in Section 3.2, the basics of the 802.11 MAC layer will be covered.

3.1. 802.11 Orthogonal Frequency Division Multiplexing Physical Layer

In this section, the basics of the 802.11 orthogonal frequency division multiplexing (OFDM) physical layer are explained. As mentioned earlier, only the basic OFDM physical layer is covered, and high-throughput and very high throughput extensions are ignored. OFDM physical layer has a maximum throughput of 54 Mb/s with 20 Mhz channel spacing. Standard also specifies "half-clocked" 10 Mhz and "quarter-clocked" 5 Mhz channel spacing.

The physical layer protocol data unit (PPDU) starts with preambles. The length of the preamble part is 4 OFDM symbols and includes 10 repetitions of the short training sequence, and after this, a double-length cyclic prefix [4] and two repetitions of the long training sequence. The short training sequence is for packet detection, automatic gain control (AGC), diversity selection, and coarse timing and frequency synchronization. The long training sequence is for channel estimation and fine timing and frequency synchronization.

[4] In 802.11 standard called Guard Interval (GI) 2

After training sequences, there is the signal field, independently encoded single BPSK modulated OFDM symbol which carries all necessary information for packet reception (or to calculate transmission time). It includes fields for packet length and MCS.

The signal is followed by the DATA field, which carries the actual payload. The whole DATA field is encoded into a single code block. It starts with the 16 bit long SERVICE field, which is in the current standard version used only to initialize the scrambler at the receiver side. This requires 7 bits, and the rest of the bits in the SERVICE field are reserved for future use. The signal, together with the SERVICE field, is called the physical layer convergence procedure (PLCP) header. The DATA field also includes the PLCP service data unit (PSDU), tail bits needed by the encoder, and possible paddings to fill an integer number of OFDM symbols.

All the bits inside of the DATA field are scrambled using a scrambler with 7 bits of memory. The initial state of these seven bits is set to a pseudo-random state before packet transmission. The scambler is defined as a shift register with feedback that is both modulo 2 added into data bits and fed back to the shift register. This kind of scrambler is also known as linear feedback shift register (LFSR). Feedback is defined to be in polynomial form $S(x) = x^7 + x^4 + 1$. The purpose of the scrambler is to randomize data before encoding to break long sequences of zero or one. This also makes the probability of zero and one close to 0.5, which is a useful property for decoding algorithms.

The 802.11 standard specifies a convolutional code using polynomials 133_8 and 171_8. This code is also known as "NASA standard code" as it was used in NASA deep space program [40]. This code is non-systematic and it has a maximum free distance for codes with a constraint length of 7 [41]. The free distance of this code is 10. The encoder for the 802.11 convolution code can be seen in Figure 8. An unpunctured code rate of the 802.11 convolutional code is 1/2, but the standard specifies puncturing patterns for rates 3/4 and 2/3.

Before modulation, the data stream is interleaved. The 802.11 standard specifies two permutations. The first one allocates adjacent bits to non-adjacent subcarriers, and the second one allocates adjacent bits alternately onto less and more significant bits. Together these permutations randomize errors over one OFDM symbol, which will decrease the probability of error bursts, and on the other hand, increase the probability of good decoding.

All the transmitted symbols excluding training sequences carry four BPSK modulated pilot subcarriers whose polarity changes pseudo-randomly, reusing the scrambling pattern used to scramble data. Pilots allow updating different estimates (time, phase, frequency offset, SNR) during the reception of the signal, which can be especially important for equipment with a low-quality oscillator having high phase noise.

802.11 OFDM physical layer supports modulation constellations BPSK, QPSK, 16-QAM, and 64-QAM. A simplified block diagram of the 802.11 physical layer can be seen in Figure 9.

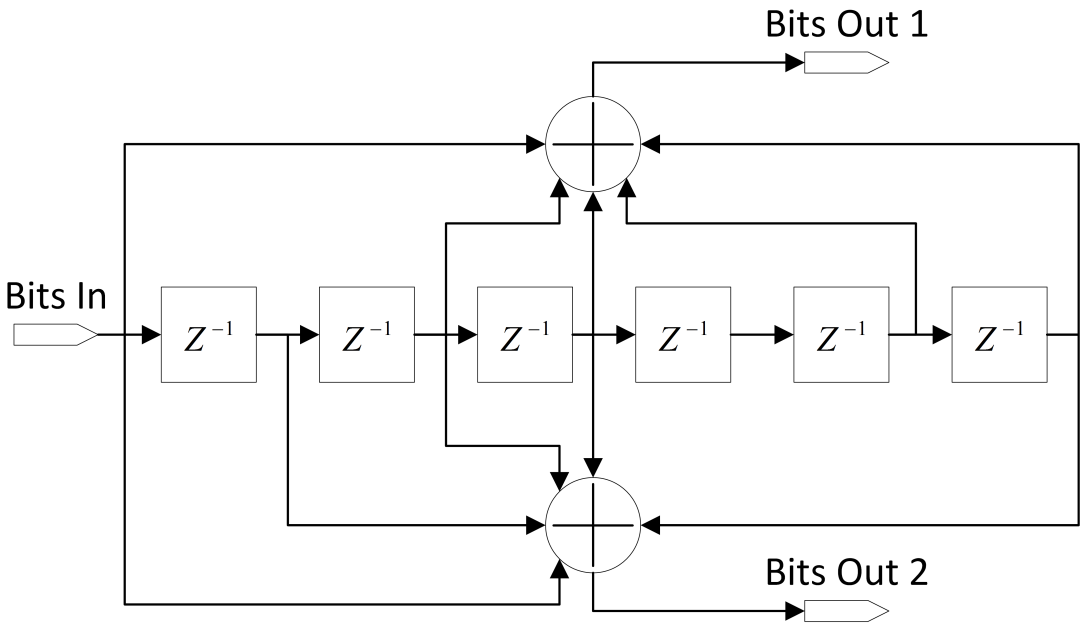

Figure 8. 802.11 OFDM Physical layer convolutional encoder

3.2. 802.11 Media Access Control Layer

The 802.11 media access control (MAC) layer is designed to cope with the challenges of unlicensed communication, where many devices are sending data without centralized control. This is mostly achieved by the distributed coordination function (DCF). DCF implements CSMA with collision avoidance (CSMA/CA) type of MAC service.

In the carrier sense multiple access (CSMA), carrier sense is used to listen to the medium before transmitting a packet. This kind of protocol is also known as listen before talk. The 802.11 DCF also specifies the virtual carrier sense which is based on the duration field carried by packets. The duration field is used to initialize a network allocation vector (NAV), a counter counting towards zero. While the counter is nonzero, the virtual carrier sense indicates a busy medium. If a transmission fails, the transmitter uses a random length back-off whose maximum possible value increases exponentially until it reaches the maximum allowed value. This is called exponential back-off. 802.11 specifies ready to send (RTS) and clear to send (CTS) packets which can be used to avoid the so-called hidden node problem.

The PSDU contains 3 main fields, MAC header, frame body, and frame check sequence (FCS). In 802.11 PSDU content changes depending on the MAC frame[5] type, subtype, and several other fields in the Frame Control field. 802.11 defines 3 frame types: control, data, and management. The 802.11 frame structure can be seen in Figure 10. In the case of data frames, the frame body usually contains a MAC service data unit (MSDU) carrying the user payload.

[5]Usually at layer 2, there are frames, whereas layer 3 has packets

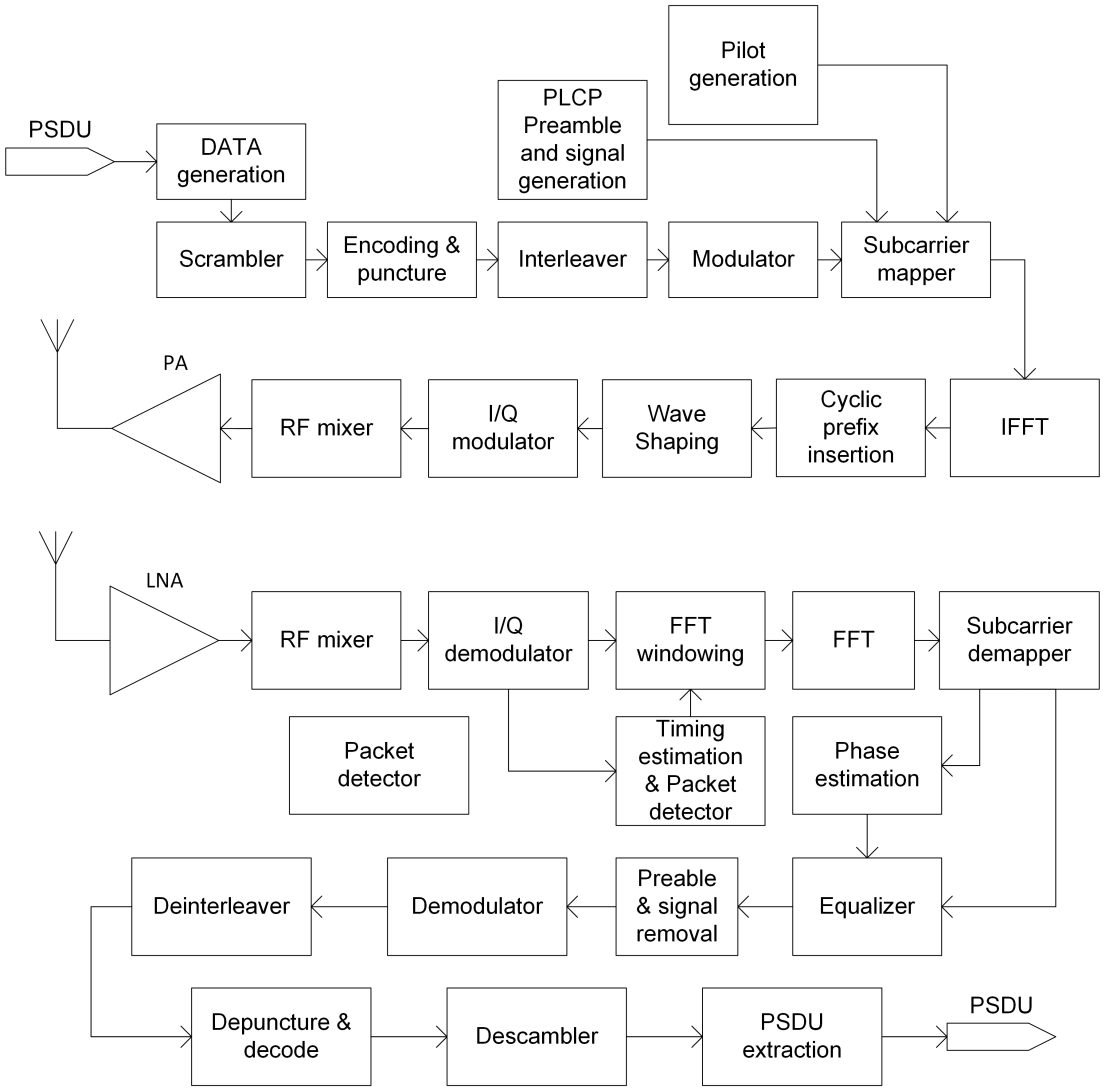

Figure 9. Block diagram of 802.11 Physical layer

Frame Control	Duration / ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	QoS Control	HT Control	Frame Body	FCS

Figure 10. 802.11 MAC frame format

4. SIMULATOR IMPLEMENTATION AND DESIGN

The simulator used in this Master's thesis is implemented using C++. The simulator is mostly based on modern C++, and it is compiled using standard version 14. For soft demapping, it utilizes Open-CL parallel computing. Some parts of the simulator are loosely based on the open-source gr-ieee802-11 project [42].

The design uses mainly standard-compliant processing blocks, although some parts which are not needed by the simulation have been omitted. For example, fast Fourier transform (FFT) and inverse FFT (IFFT) are not used in the simulation. Instead, the communication channel is implemented in the frequency domain. Used channels are constant for each packet copy and channel type is either AWGN or Rayleigh. Implementing channels in the frequency domain saves calculation resources. The result is the same as with white noise inserted in the frequency domain because noise distribution remains the same [43 p. 501].

Two different simulators have been implemented for this thesis, one for joint decoding over multiple packet copies and one for EXIT analysis. Both simulators have a similar structure, higher-level controlling code (main loop), packet generator, and decoder. Packet generator, decoder, and many related support functionalities were implemented in a separate shared library called libdecode. The simulator is largely the same as used in [4] for 802.11 simulations. However, it was refactored to work independent from GnuRadio, added functionalities for traceability (decoding trajectory), functionality for EXIT analysis, and support for Rescue code.

Design goals of the simulator are to add needed support for 802.11 OFDM packet joint decoding between packet copies while keeping as much as possible standard compliance and comparison between 802.11 convolutional code against serial concatenated convolutional code used in RESCUE project. Joint decoding between packet copies requires packet payloads to be differently interleaved between packet copies to work efficiently[6]. Therefore in the simulator, an additional random interleaver has been added before convolutional encoding. To avoid unnecessary changes from the standard, interleaver seed is selected based on scrambler seed and therefore does not require any additional data to be sent. Furthermore, as the design goal is to keep as much as possible standard backward compatibility, the simulator has support to interleave only MSDU, keeping the header and FCS in their original position. This option is called backward compatibility mode in this thesis.

Because usually the receiver determines received packet goodness based on FCS, in the simulator, FCS is always calculated for non-interleaved PSDU. This principle allows all packet copies to carry the same FCS and combining of FCS LLRs. The bad side of this design principle is that even in backward compatibility mode (only MSDU interleaved) FCS check would fail on a standard compatible receiver.

The simulator uses 3 different interleavers in the encoding/decoding process. Interleaver $\pi_{1,n}$ is added before scrambling the payload to enable joint decoding, π_2 is used together with inner code to separate it from the outer code, and π_3 is interleaver from the 802.11 standard used between coding and mapping symbols to IQ samples having a size of one OFDM symbol.

[6]Correlated interleavers / no interleaver may cause unexpected *a posteriori* information loop between decoders, making decoder *a priori* input from $L_e(u)$ to approach $L_p(u)$ due to leaked information from adjacent bits

The simulator has two outer convolutional codes supported as outer code: 802.11 non-recursive non-systematic code having octal polynomials $(133_8, 171_8)$ and systematic non-recursive convolution code from Rescue with polynomials $(7_8, 5_8)$. Outer puncture is based on 802.11 puncture patterns. Inner code in the simulator is the doped accumulator used in the rescue code. The doped accumulator is modeled here with systematic recursive convolutional code with puncturing. All codes used in the simulator are terminated. Because the simulator has both recursive and non-recursive convolutional codes, termination is trellis aided meaning the encoder will select the input bit that generates zero into the first memory stage during the termination process.

The simulator supports modulations defined in the 802.11 standard for OFDM physical layer. Supported modulations (and code rates) are BPSK 1/2, BPSK 3/4, QPSK 1/2, QPSK 3/4, 16-QAM 1/2, 16-QAM 3/4, 64-QAM 2/3, and 64-QAM 3/4.

Simulator main parts are described in the following sections. In the simulator, packets are called either TxPacket or RxPacket, depending on whether the packet is before or after the channel.

4.1. Packet Generator Implementation

The packet generator creates TxPacket from MSDU (see Section 3.1) based on given parameters. The same generator implementation is used for different simulation cases with the 802.11 and the RESCUE.

Overall packet generator design can be seen in Figure 11. Although the figure shows the logical flow of packet generation, the actual result called TxPacket contains not only samples out but also all intermediate packet generation results that are useful for EXIT analysis and help in debugging. Some of the packet generation steps may not be applied depending on the configuration.

For joint decoding, a random interleaver $\pi_{1,n}$ is added before scrambling, and depending on the simulated case, it covers either the whole PSDU or in backward compatibility mode only MSDU. The bits not covered by the interleaver will be passed through as it. The service field is never interleaved as it is used to initialize the scambler and determine the interleaver seed.

In the case of the RESCUE code, interleaver π_2, inner encoder, and inner puncture are used. Otherwise, those blocks are bypassed. Inner encoder and inner puncture together form a doped accumulator used in the RESCUE code.

4.2. Channel Implementation

Channel models used in this thesis are AWGN and Rayleigh. Because both of these channels are similar, differing only on channel gain, the same implementation is used for both. For the Rayleigh channel, coherence time is assumed to be longer than transmit time of a single packet copy. Therefore the Rayleigh channel is modeled with AWGN where gain changes between packet copies.

Channel takes in a TxPacket and outputs a RxPacket. To allow certain functionalities on the decoder side, the RxPacket has not only the channel bits but also some side information related to packet length, MCS, and scrambler seed.

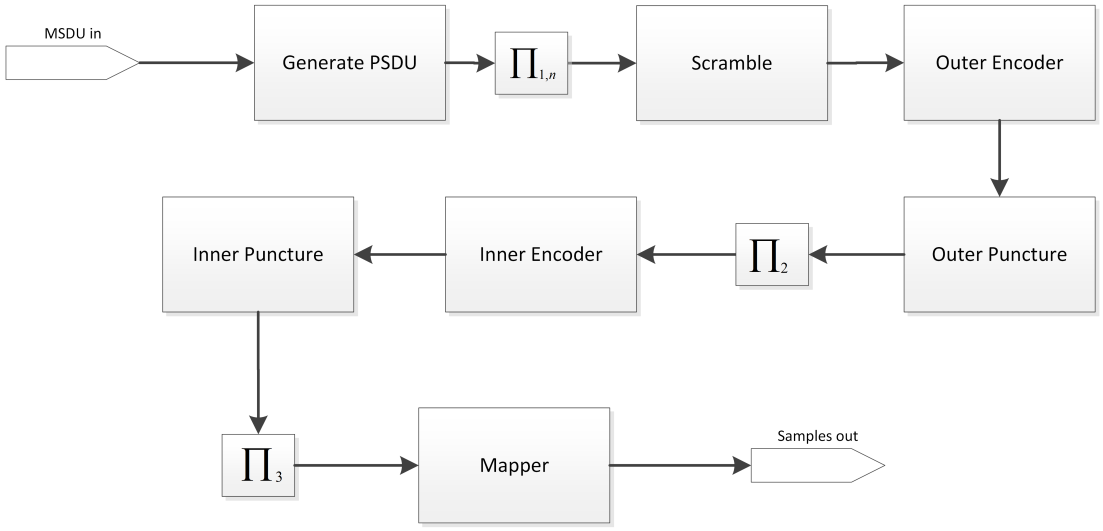

Figure 11. Simulator encoder block diagram

4.3. Decoder Implementation

The decoder supports similar configurability as Packet Generator plus the possibility to ignore certain decoding errors, which would normally stop the decoding process. These errors are namely signal, scrambler, and header with some bits wrongly decoded. This is to allow studying not only the performance of the system as a whole but also the performance of the payload decoding in an ideal setup. For example, in case of scrambler and signal errors are ignored, the decoding process will continue even with erroneous decoding and substitute these parts with ideal information received as side information in the RxPacket (see Section 4.2). The decoder also supports iterative demapping, where the demapper is taken as part of an iterative decoding process.

The joint decoding process of multiple packet copies runs mainly on parallel threads except for demapping, which runs only a single instance at the time (see details in Section 4.3.1). Decoder main logic is depicted in Figure 12.

The decoder uses the soft demapper and the Log Map decoder as part of the decoding process. These are described in Section 4.3.1 and Section 4.3.2 correspondingly.

4.3.1. Soft Demapper

Soft demapper is implemented using C++ and OpenCL. The actual demapper implementation is done in OpenCL, which allows easy parallel execution. C++ implementation includes controlling code that opens OpenCL context, calls the actual OpenCL implementation, and provides interfaces for the rest of the decoder. Demapper has separate OpenCL implementations for BPSK, gray coded QPSK, general gray coded constellation. There is also a separate implementation that supports any time-invariant constellation having an integer amount of mapped bits. OpenCL is a C-like programming language that can be run on many platforms, including graphics processing units (GPU) to central processing units (CPU) in a parallel manner.

Because demapper uses pre-allocated resources (buffers etc.), it is not thread-safe and therefore protected by mutexes. Although concurrency is not allowed in demapping execution between packet copies, there is a significant gain from parallel processing in terms of speed depending on GPU / CPU capabilities where OpenCL code is run.

Soft demapper supports *a priori* information which is a requirement in iterative decoding. Implementation is based on the theory presented in Section 2.2.

4.3.2. Log-Map Decoder

The Log-Map decoder is the most resource-hungry processing block in the simulator due to its complex processing. Log-Map decoder's theoretical background is presented earlier in Section 2.7.

4.4. Simulator for Iterative Decoding

Iterative decoding simulation combines earlier presented packet generator, channel, decoder, and control logic to run a simulation and collect needed results. The simulator provides required binding for command line parameters to allow running different scenarios without changing the code itself.

The simulator reports signal error ratio (SER), scrambler error ratio (SCER), header error ratio (HER), frame error ratio (FER), and bit error ratio (BER) per SNR point in a format that is directly readable by Matlab. The decoder provides information about scrambler and signal goodness, but all the other outputs are calculated in the control code. Header and frame (PSDU) are counted as erroneous in case there are any bits with the wrong value. The simulator design for iterative decoding is depicted in Figure 13.

4.5. Simulator for Extrinsic Information Transfer Analysis

The simulator for EXIT analysis uses primarily the same building blocks as the simulator for iterative decoding. In addition, it has a block creating needed *a priori* information for the decoder and a block estimating outcoming extrinsic information. *A priori* information generation and extrinsic information estimation is described in Section 2.9.

The decoder block works a bit differently in the case of EXIT analysis than in iterative decoding simulations. The decoder structure seen in Figure 12 is cut from a suitable place, and *a priori* information is fed into the decoding process. For example, EXIT analysis, including outer code, inner code, and demapper, decoding is not done between packets. Instead, *a priori* information is fed from the point where *a priori* information from another packet copy comes in. Simulator reports results in a Matlab compatible format, including extrinsic information, BER, *a posteriori* information, and *a posteriori* BER per packet as a function of *a priori* information. Simulator for EXIT analysis is depicted in Figure 14.

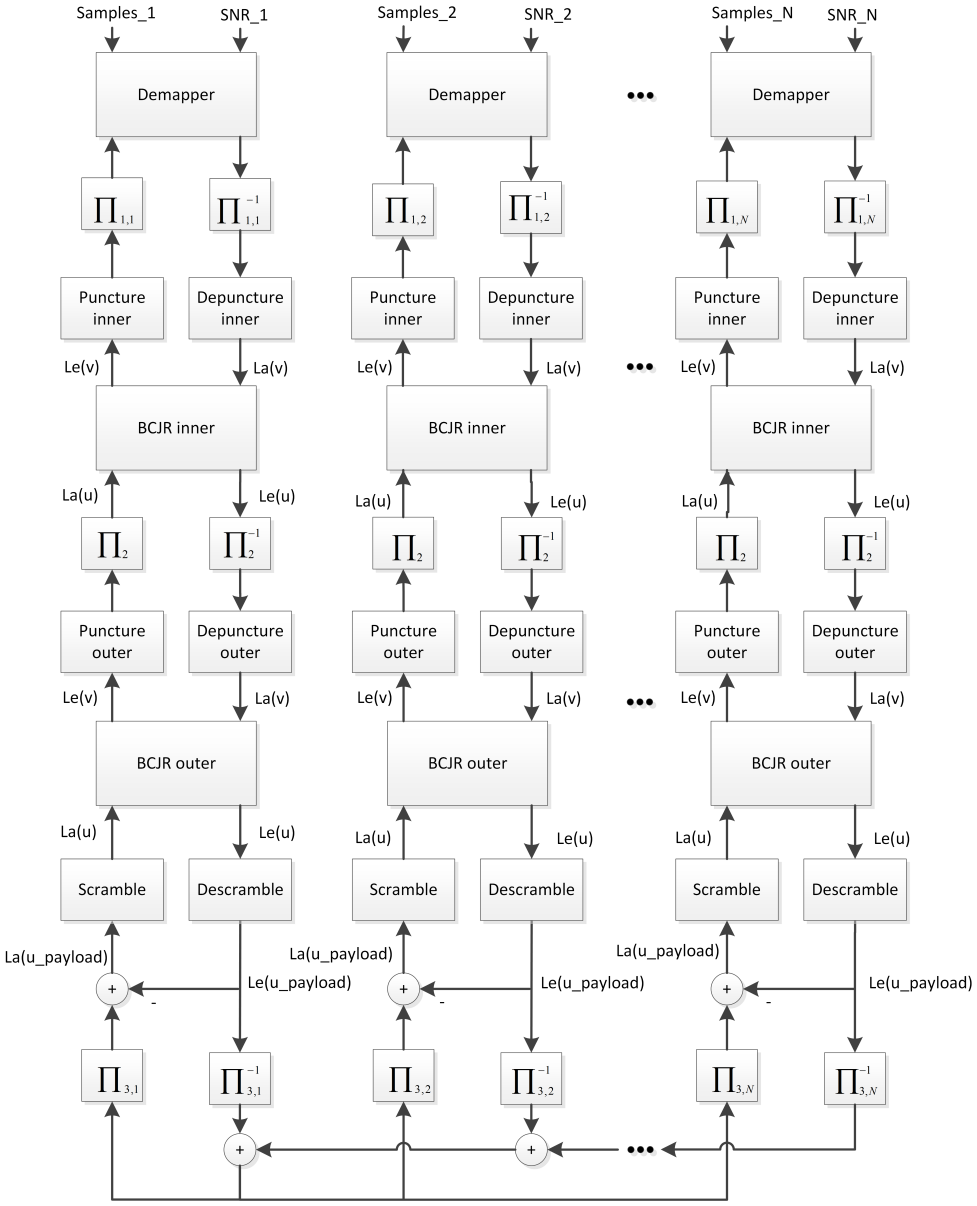

Figure 12. Simulator decoder block diagram

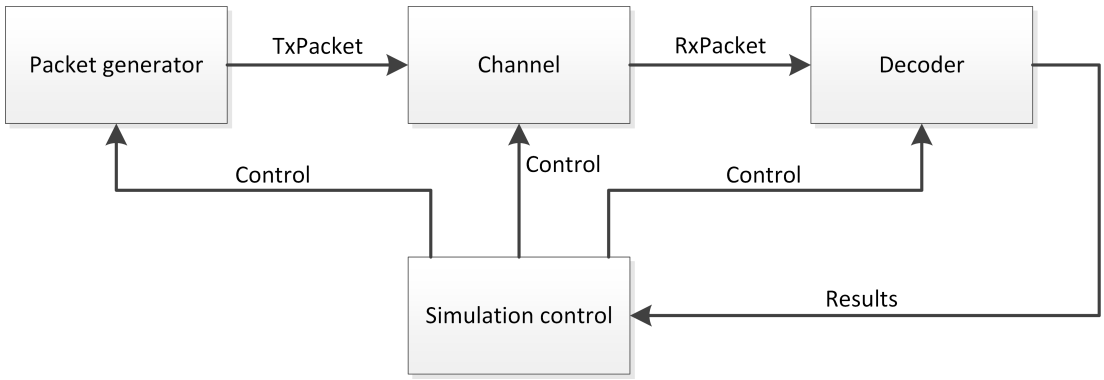

Figure 13. Simulator for iterative decoding

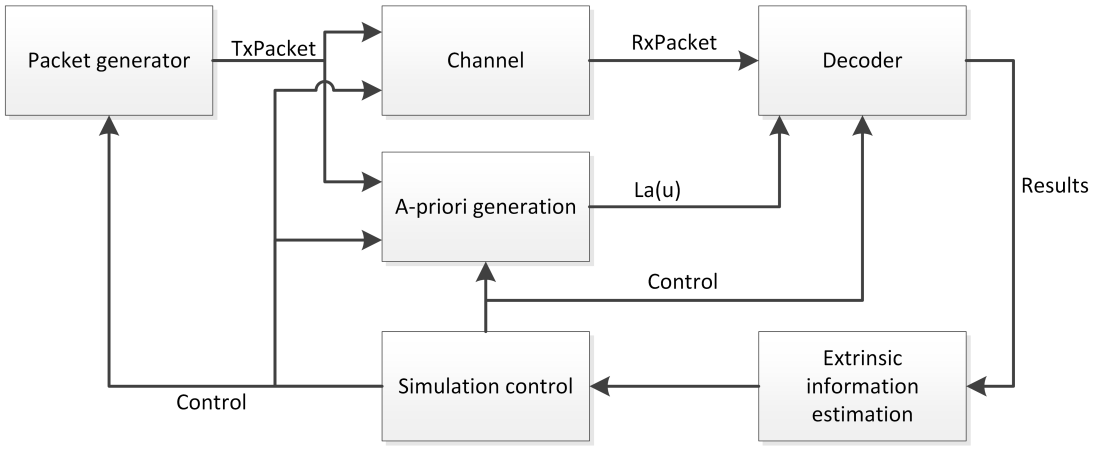

Figure 14. Simulator for EXIT analysis

5. EXTRINSIC INFORMATION TRANSFER ANALYSIS OF 802.11 CONVOLUTION CODE AND RESCUE CODE

In this chapter, the results of the EXIT analysis are presented. The structure of this chapter is as follows: First, in Section 5.1, EXIT analysis results of the 802.11 code are studied, then in Section 5.2, EXIT analysis results of the RESCUE code are presented. Later in Section 5.4, demapper feedback with the 802.11 and the RESCUE code is studied. As the demapper loop is examined separately, all other simulations in this chapter are without demapper feedback.

5.1. Extrinsic Information Transfer Analysis of 802.11 Convolutional Code

In this section, EXIT analysis results with the 802.11 convolutional code are presented. Results are depicted in charts showing extrinsic information transfer between two decoders, one on the horizontal axis and another one on the vertical axis. Each figure shows results for selected MCS:s and there is a separate figure for each selected SNR point. SNR points are 0 dB, 3 dB, 6 dB and 10 dB. Results can be seen in Figure 15. Considerable detail in exit curves is that they are crossing slightly before mutual information 1, which predicts some error floor.

5.2. Extrinsic Information Transfer Analysis of RESCUE Convolutional Code

Analysis for the RESCUE code is done similarly as for the 802.11. All the plots use the same MCS and SNR values to allow easier comparison between codes. Results can be seen in Figure 16. In this simulation, there are eight iterations between the inner and outer code.

As the RESCUE code also has an inner code, doped accumulator, results for it can be seen in a separate Figure 17. Because inner code's puncturing is always the same, there is only one selected MCS and several SNR points. For comparison, the figure also includes lines for demapper output information without inner code. As can be seen, a doped accumulator is beneficial when there is *a priori* information available. This can mean, for example, iterative coding between inner and outer code. On the other hand, without *a priori* information, a doped accumulator does not bring any benefit, and in fact, it outputs less information than it gets in from the demapper.

In EXIT analysis results with full RESCUE code (inner and outer code) it can be seen that with higher *a priori* information all scatters tend to saturate to extrinsic information output 1. This is due to iterative decoding between inner and outer code. It can be understood that incoming *a priori* information pushes the decoder over the turbo cliff.

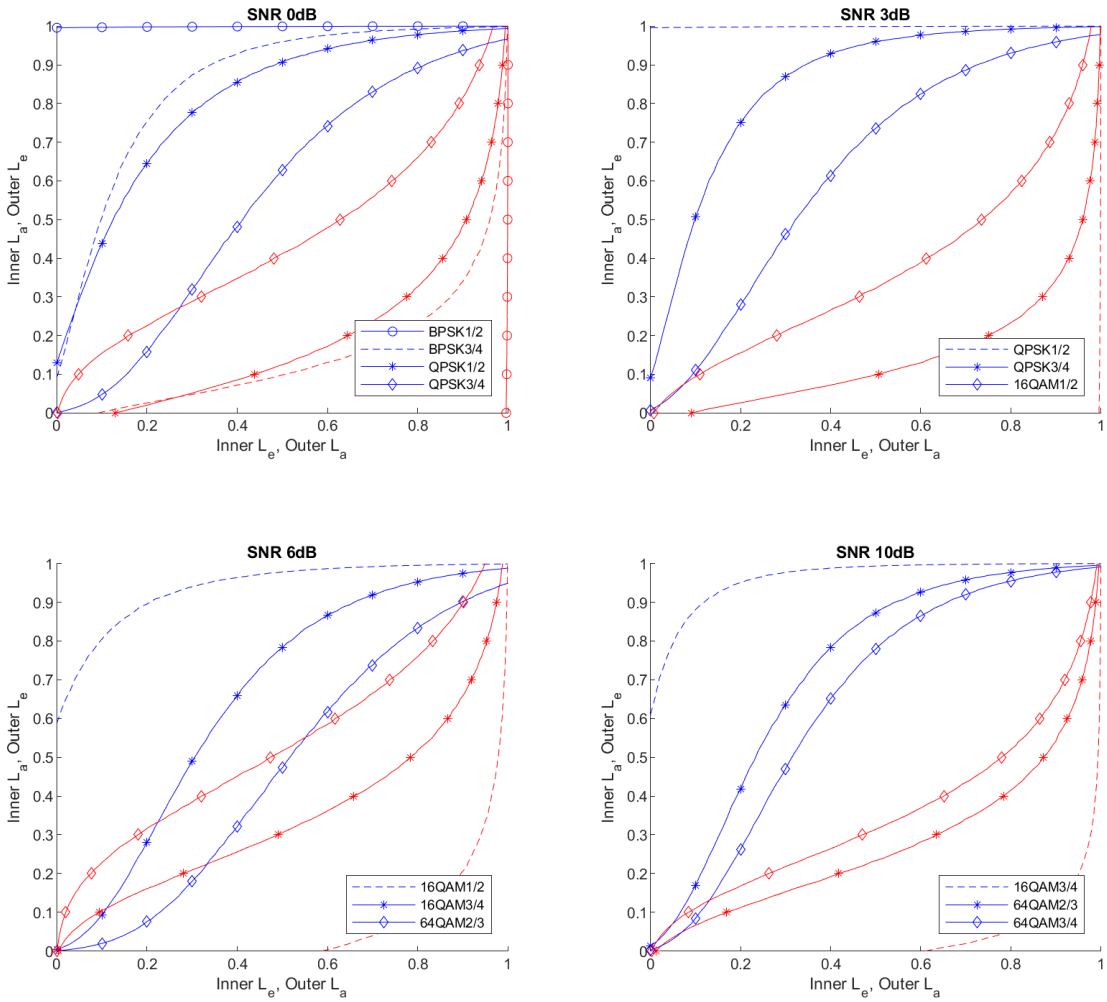

Figure 15. EXIT analysis results for the 802.11 code. Red lines are for inner decoder, blue for outer.

5.3. Comparison between RESCUE and 802.11 Extrinsic Information Transfer Results

In this section, the RESCUE code and the 802.11 code are compared in means of extrinsic information transfer. As comparison is for codes, not modulation, there is one plot per puncturing pattern. As none of the modulations support all the puncturing patterns (code rates), only selected SNR & MCS pairs are shown. In simulations, two SNR regions are considered, low and medium. These ranges are defined through the 802.11 code. At the low range, the 802.11 code outputs close to zero bits per extrinsic information bit and at the medium range around 0.5 bits. High SNR range is not considered here. With high SNR, packets are typically decoded without errors and iterative processing does not bring additional gains. Modulations used here are either 16-QAM or 64-QAM, but SNR is scaled to fulfill the aforementioned ranges, and therefore, modulation itself is irrelevant here.

Results can be seen in Figure 18. As can be seen, at low SNR the RESCUE code outputs slightly more information without *a priori* input than the 802.11 code, which is beneficial in iterative decoding at the low SNR region. When SNR gets higher, the

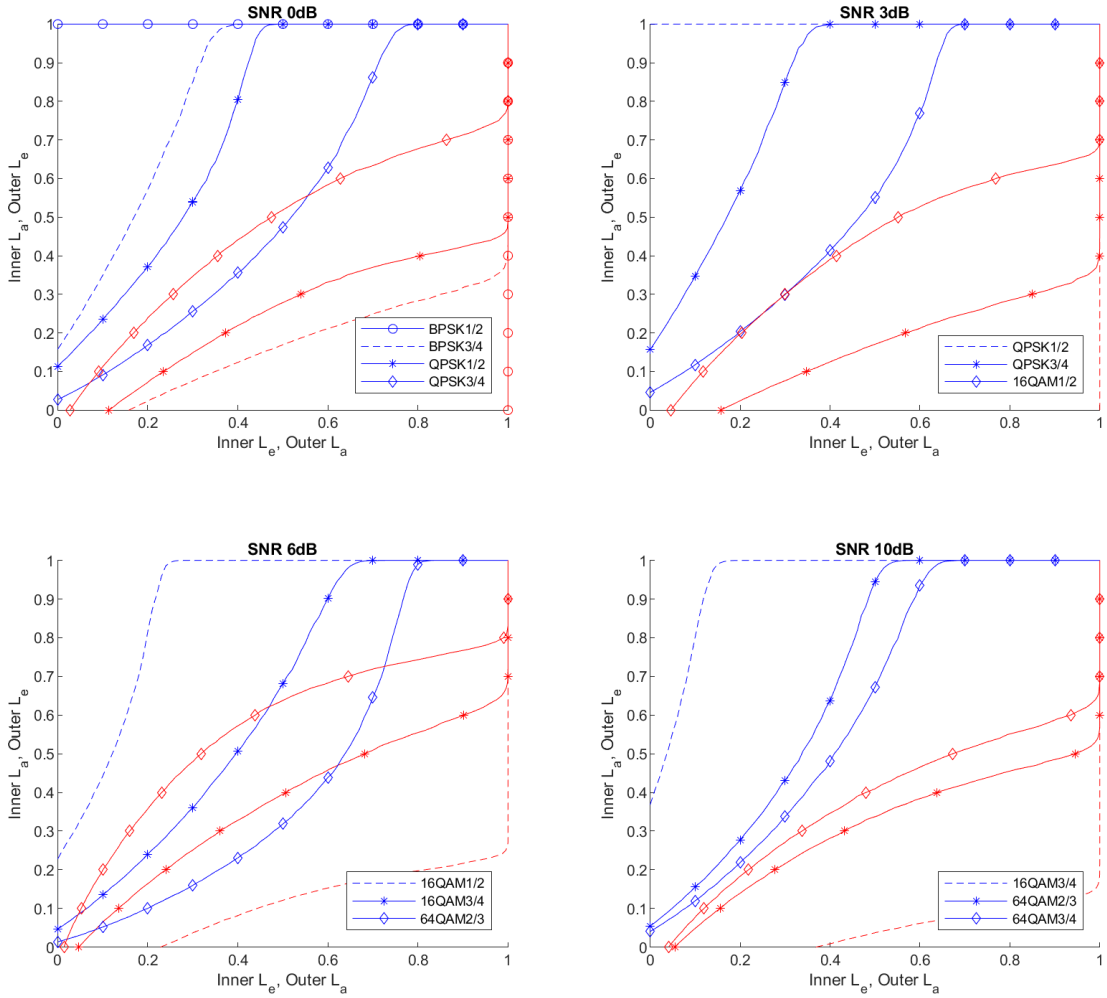

Figure 16. EXIT analysis results for the RESCUE code. Red lines are for inner decoder, blue for outer.

802.11 code begins to output more information, but in both cases, the RESCUE code performs better when the amount of *a priori* information gets higher. This result is expected as the RESCUE code is optimized for iterative decoding, whereas the 802.11 code should allow the error-free reception of a single packet.

5.4. Extrinsic Information Transfer Analysis with Demapper Loop

In earlier research, iterative demapping is deemed to be ineffective in the case of Gray coding, which is the case with the 802.11 physical layer. Results with and without demapper loop from EXIT simulations using 64-QAM can be seen in Figure 19 and Figure 20.

The results show that there is a significant gain from the demapper loop with lower SNR, especially when using the RESCUE code. However, at the higher SNR region, this effect seems to disappear, and it looks like the demapper loop may even harm. Simulations showed a similar effect also for the smaller 16-QAM constellation, although at the lower SNR range. This might be due to the fact that at low SNR, there

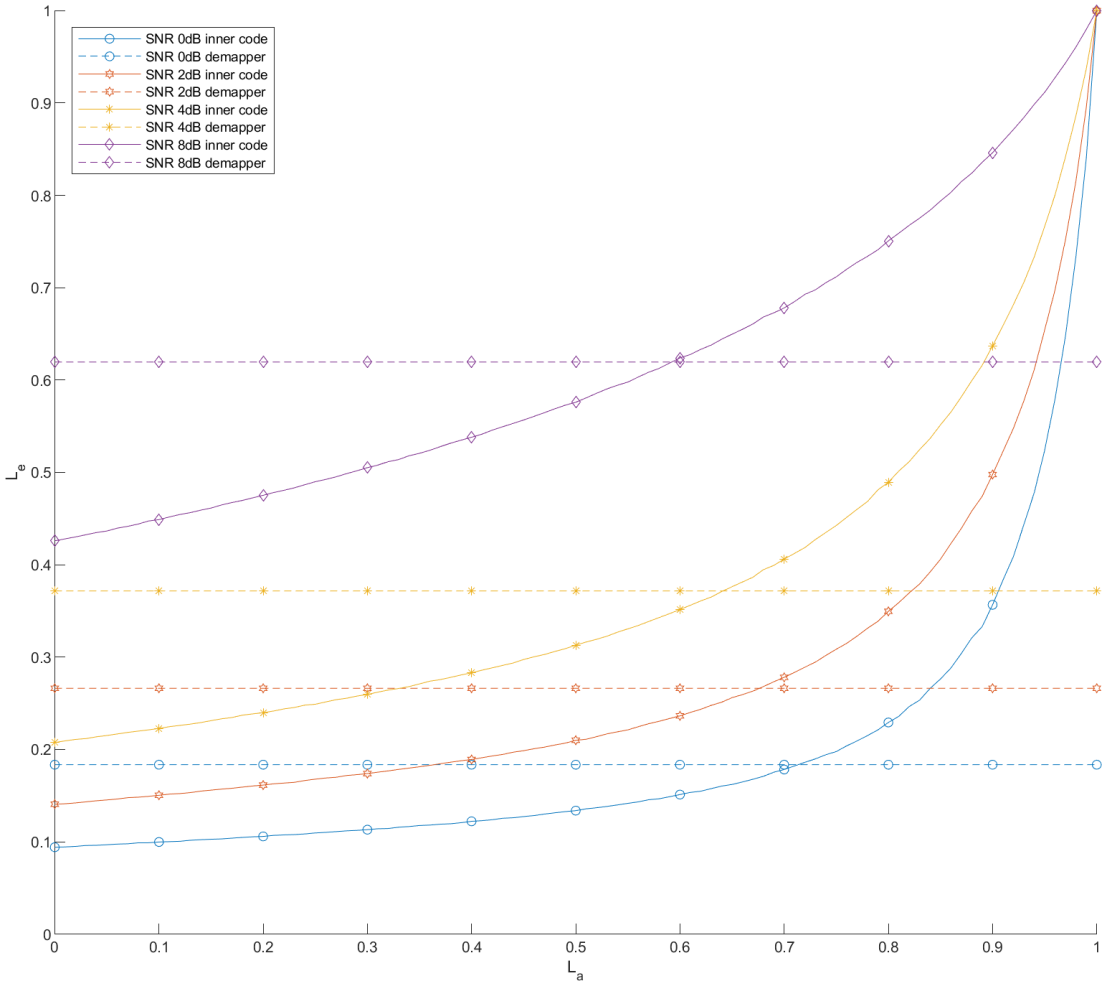

Figure 17. EXIT analysis results for the RESCUE inner code

are more symbols with multiple bit errors, and therefore, *a priori* information has a bigger positive effect.

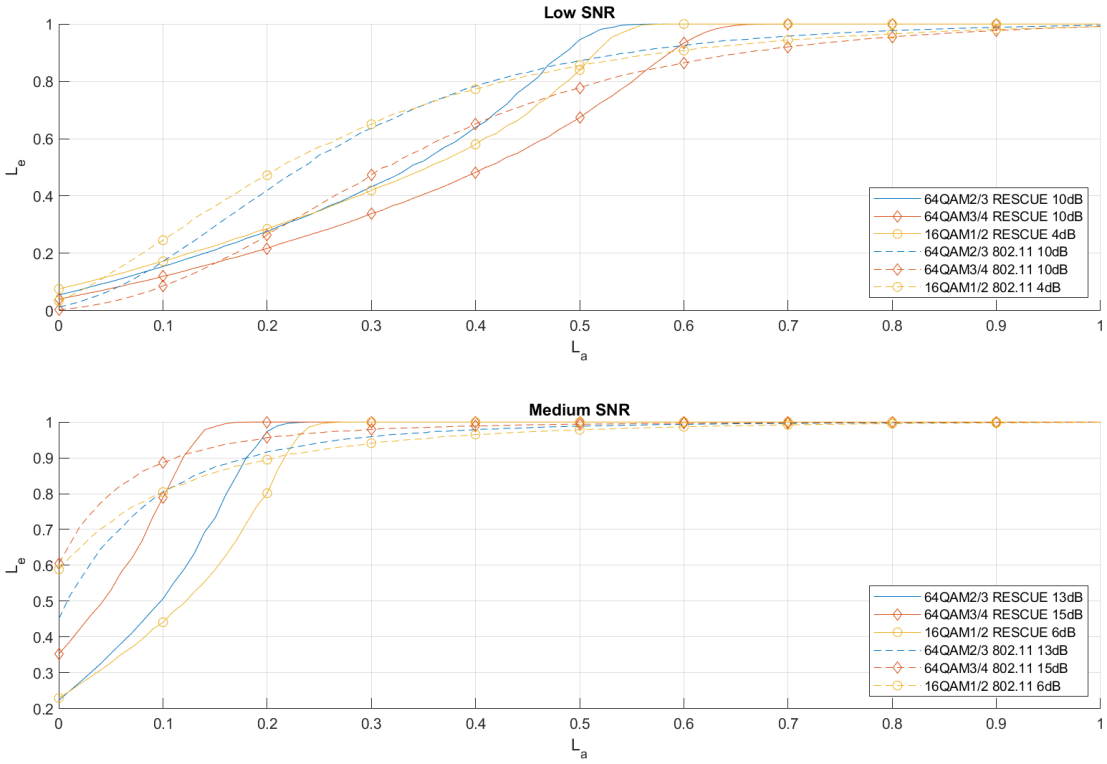

Figure 18. EXIT analysis results for RESCUE and 802.11

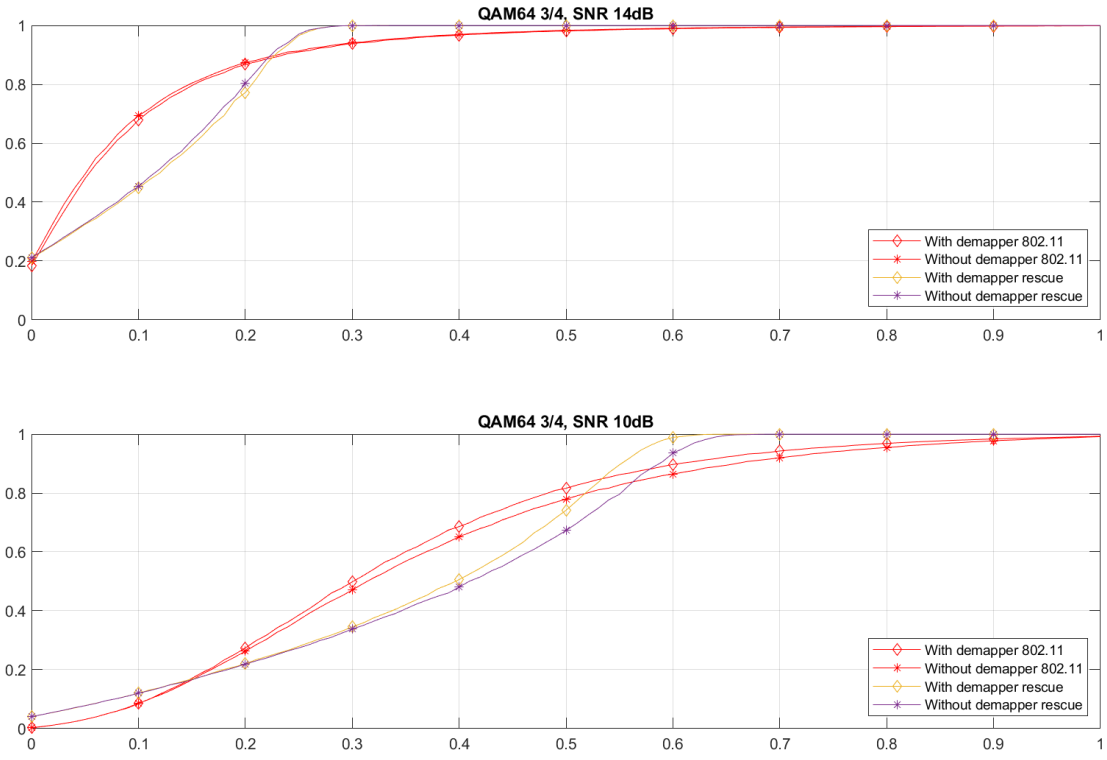

Figure 19. Simulator for EXIT analysis

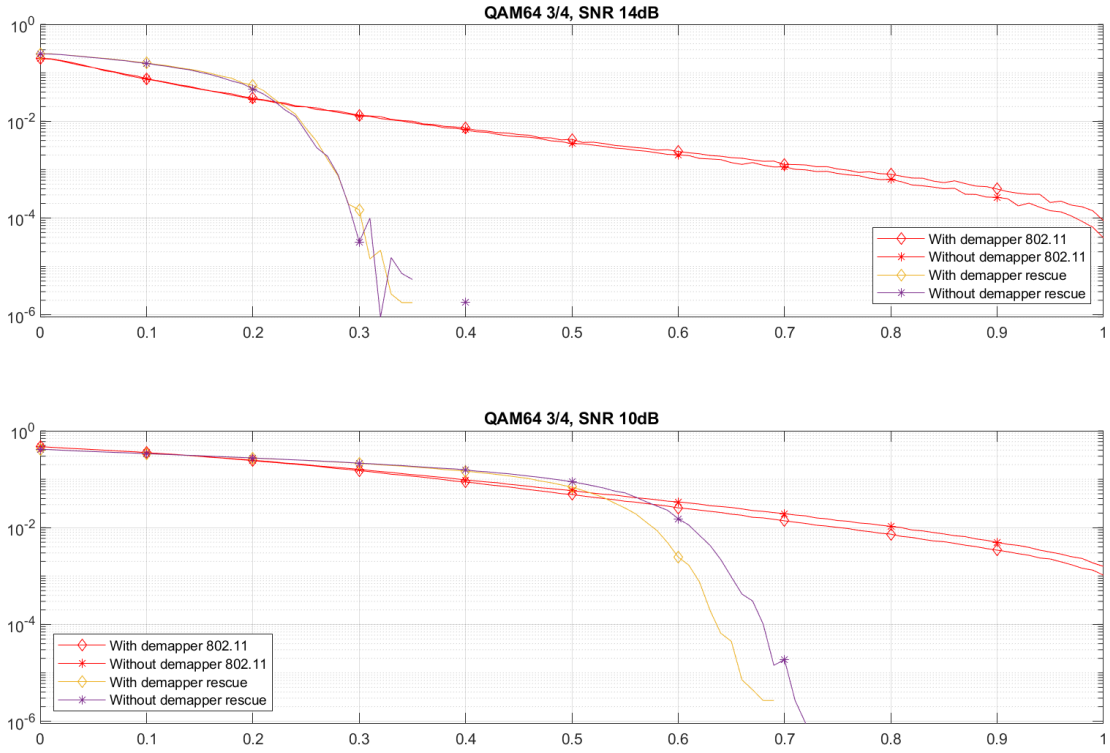

Figure 20. Simulator for EXIT analysis

6. SIMULATION RESULTS OF PROPOSED DECODING SCHEME

In this chapter, simulation results from the simulator are introduced. The chapter is divided into three main parts, results with the 802.11 code, with the RESCUE code, and finally, a comparison between these two. In simulations, a maximum of 5 copies per packet is sent over the AWGN or the Rayleigh channel and then jointly decoded.

For both codes, RESCUE and 802.11, there are similarly formatted figures showing BER, FER, HER, SCER, and SER. BER and FER are shown separately for each number of packet copies, whereas HER, SCER, and SER are per copy as they can be seen as blocking errors in decoding. The header is needed to detect whether or not the packet should be added into the joint decoding queue, signal errors affect to detection of the MCS, and scrambler errors into the descrambling process. In simulations, all errors are ignored, and in the case of a signal error the correct MCS is used anyway in the decoding, and in the case of a scrambler error, the scrambler is initialized with a known index. Because scatters fell so steeply in many cases that it might be hard to see the actual behavior, for illustration purposes, all zero samples in FER figures are replaced with 2.22×10^{-16}.

The decoding performance of both codes is compared against Maximum Ratio Combining (MRC), which can combine received samples so that SNR after the combining is maximized. Although MRC cases were not simulated, theoretical decoding performance is depicted in some of the figures. MRC gain for AWGN is calculated $10 \log_{10}(N)$, where N is the number of copies. For the Rayleigh channel, theoretical MRC performance is calculated by numerical integration of single copy AWGN results using PDF from [20 p. 211].

6.1. 802.11 Code

As the 802.11 standard defines many different MCSs, only some selected ones are covered here. BPSK with code rate 1/2 is studied because of its potential coverage gain when used with joint decoding. With the 64-QAM, code rates 2/3 and 3/4 are selected as they provide high throughput. This way, all available code rates are covered, which is more critical than modulations when investigating the performance of forward error correction. The SNR range for each simulation is selected so that the turbo cliff is centered and some area is covered from both sides. First, in Section 6.1.1, simulation results with the AWGN channel are presented, Section 6.1.2 presents results with the AWGN channel using backward compatibility mode. Finally, results with the Rayleigh channel can be found in Section 6.1.3.

6.1.1. Simulations with the Additive White Gaussian Noise Channel

Simulation results over the AWGN channel for 64-QAM with code rate 3/4 can be seen in Figure 21 and for code rate 2/3 in Figure 22. Results for BPSK with code rate 1/2 are depicted in Figure 23. A comparison of decoding performance between joint decoding and MRC is depicted in Figure 24.

From the results, it can be seen that joint decoding provides more gain than applying Maximum Ratio Combining (MRC) between samples for all simulated re-transmission numbers. For code rates 3/4 and 2/3, gains are clearly bigger, between 4.16 dB and 2.4 dB, but gain from joint decoding increases, depending on the FER target, only on the first and second re-transmission. After three packet copies, it would be better to send more energy for earlier copies. A similar conclusion can be drawn for code rate 1/2, but gains are much smaller, less than one decibel.

If header and scrambler errors were not ignored, they would be a severe limiting factor for all of the simulated cases. Even in the range where the second copy provides gains, erroneous headers and scrambler indexes would cause lots of packets to be discarded. At the low SNR region, also signal decoding becomes a limiting factor.

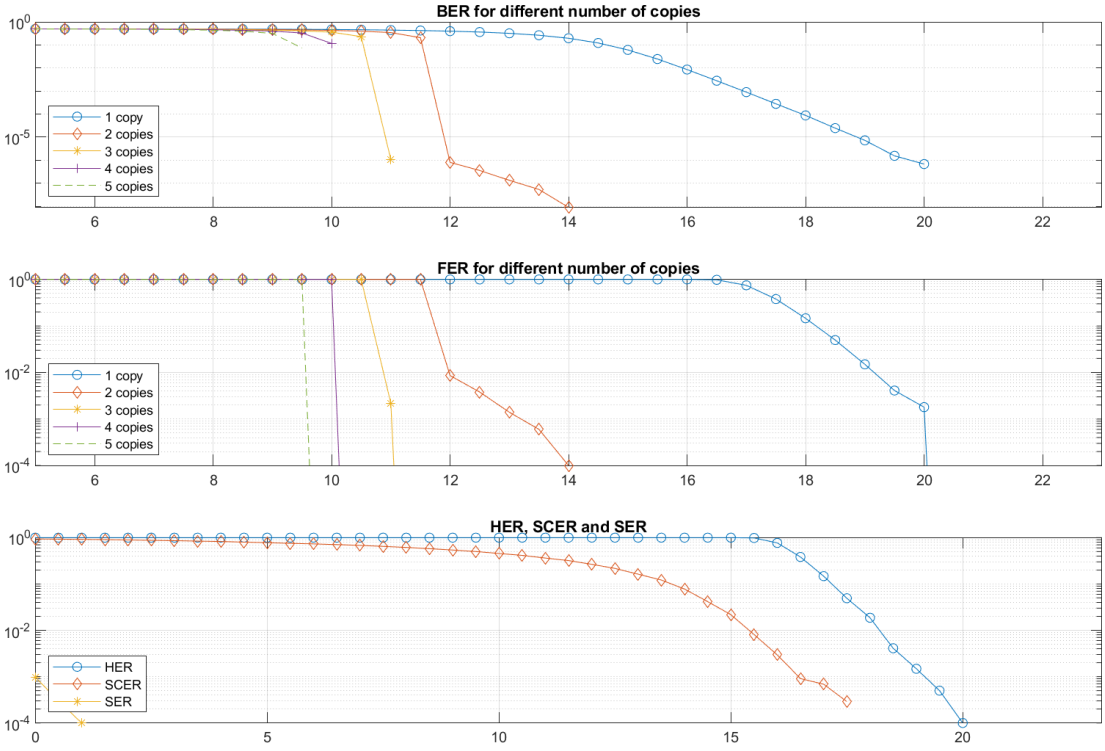

Figure 21. Simulation results for 64-QAM 3/4 with 802.11 code

6.1.2. Simulations with Backward Compatibility Mode over the Additive White Gaussian Noise Channel

Adding an interleaver for the payload breaks the frame structure where the header is transmitted first and FCS last. Therefore, an option where interleaver is limited to cover only MSDU, not the header and FCS parts, is simulated. Similarly as in Section 6.1.1, simulations for BPSK 1/2, 64-QAM 2/3, and 64-QAM 3/4 are executed. Two different cases are simulated: The Header and FCS parts combined after the last iteration or joint decoding including header and FCS parts. Joint decoding without interleaving may cause the decoder's own *a posteriori* information to propagate back to the decoder

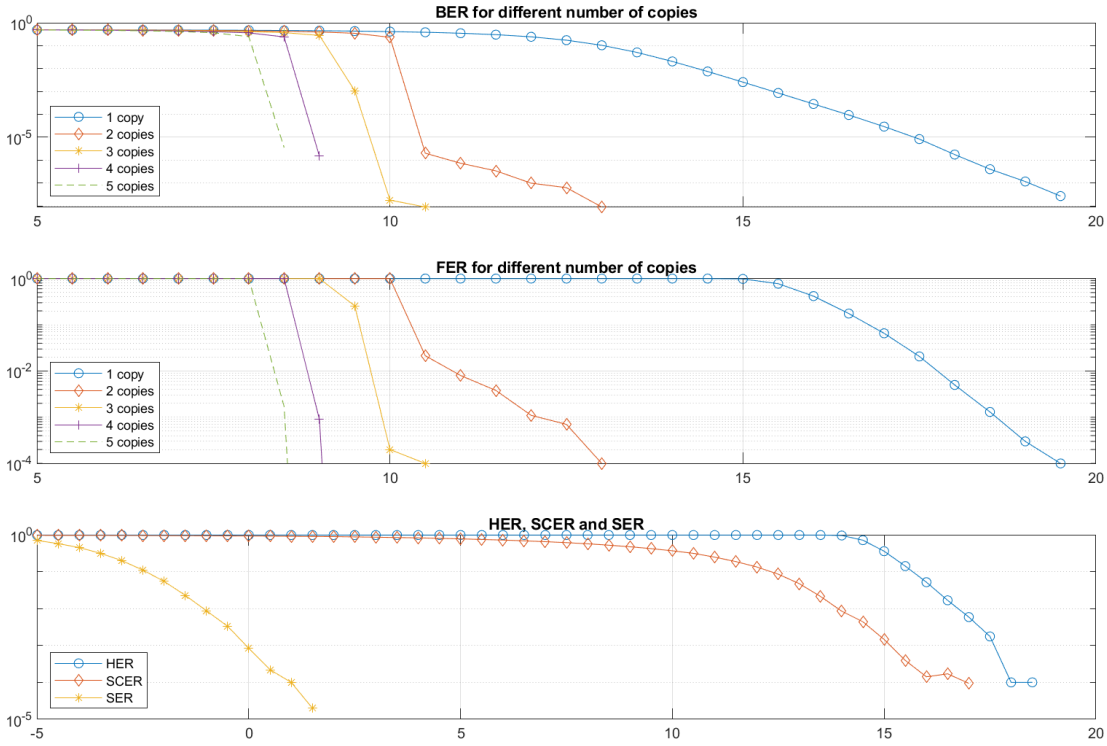

Figure 22. Simulation results for 64-QAM 2/3 with 802.11 code

through other decoders due to coupling between adjacent bits, and therefore, reduce the performance of decoding.

From simulation results in Figure 25, Figure 26, and Figure 27, it can be seen that gain from extra packet copies is much smaller than in Section 6.1.1, but in most of the cases better for first re-transmission than just sending more power. Also, it is visible that joint decoding the header and the FCS parts is a better option on lower SNR. When SNR gets higher, iterative decoding is a worse option than combining soft bits after decoding.

6.1.3. Simulations over the Rayleigh Channel

Simulation results over the Rayleigh channel can be seen in Figure 28, Figure 29, and Figure 30. A comparison of 802.11 joint decoding against MRC can be seen in Figure 31.

In simulation results, similar behavior can be seen as over the AWGN channel, gains are much higher with higher code rates. When compared against MRC, performance with higher code rates is clearly better with joint decoding, giving around 2.5 dB more gain than MRC. With rate 1/2, gain against MRC is much smaller, only around 1 dB with two copies and 0 dB with five copies.

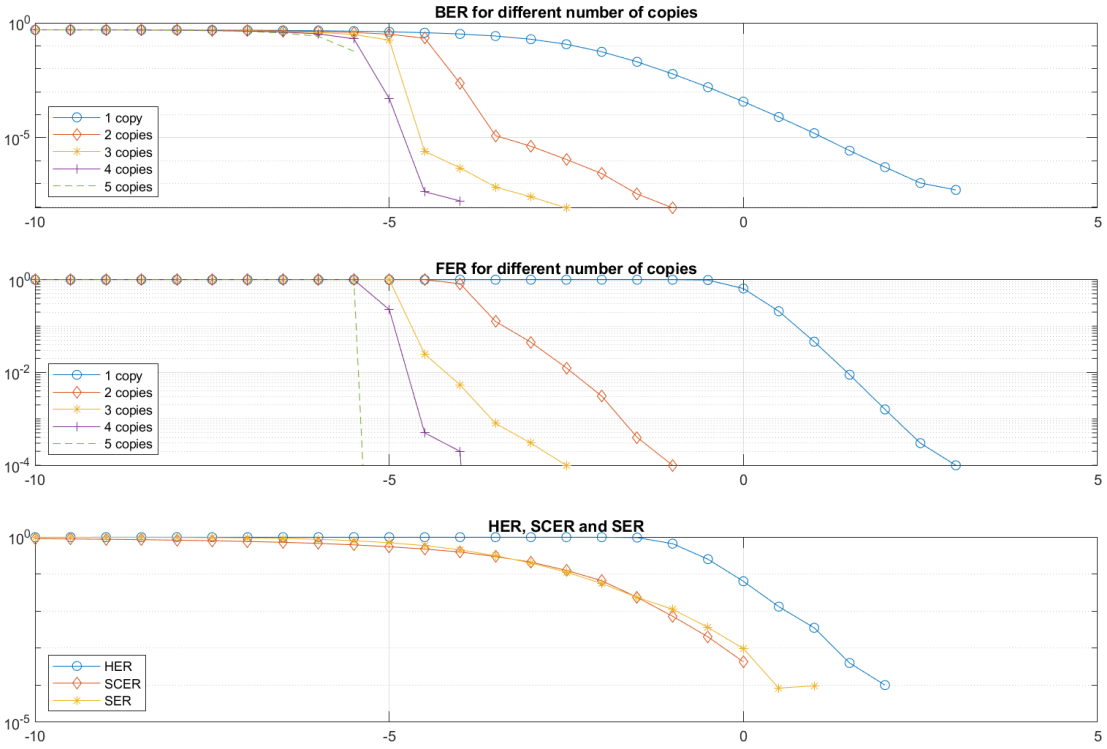

Figure 23. Simulation results for BPSK 1/2 with 802.11 code

6.2. RESCUE Code

In this chapter, simulation results using the RESCUE code are described. All the same simulations are run as in Section 6.1 except backward compatibility mode, which is not defined for the RESCUE code.

6.2.1. Simulations over the Additive White Gaussian Noise Channel

Simulation results over the AWGN channel for 64-QAM 3/4 can be seen in Figure 32, 64-QAM 2/3 in Figure 33, and BPSK 1/2 in Figure 34. A comparison between MRC and joint decoding with the RESCUE code can be found in Figure 35.

From the results, it is clear that joint decoding provides gains for other code rates except 1/2. For 1/2 rate, the first copy is already 0.5 dB worse than MRC, and therefore, instead of joint decoding, it would be a better option to send the same samples again and combine them before decoding. For code rates 3/4 and 2/3, joint decoding provides gains that are increasing with the number of copies sent. These gains are variating for rate 3/4 between 1.9 dB and 3.7 dB and for rate 2/3 between 1.3 dB and 2.5 dB.

As for the 802.11 code, scambler and header decoding are severe limiting factors. Similarly, signal decoding limits the performance at the lower SNR region. Moreover, as the signal is protected only by outer code due to small block size, it is visible that there are more decoding errors also on higher SNR region because the RESCUE outer code is relatively weak.

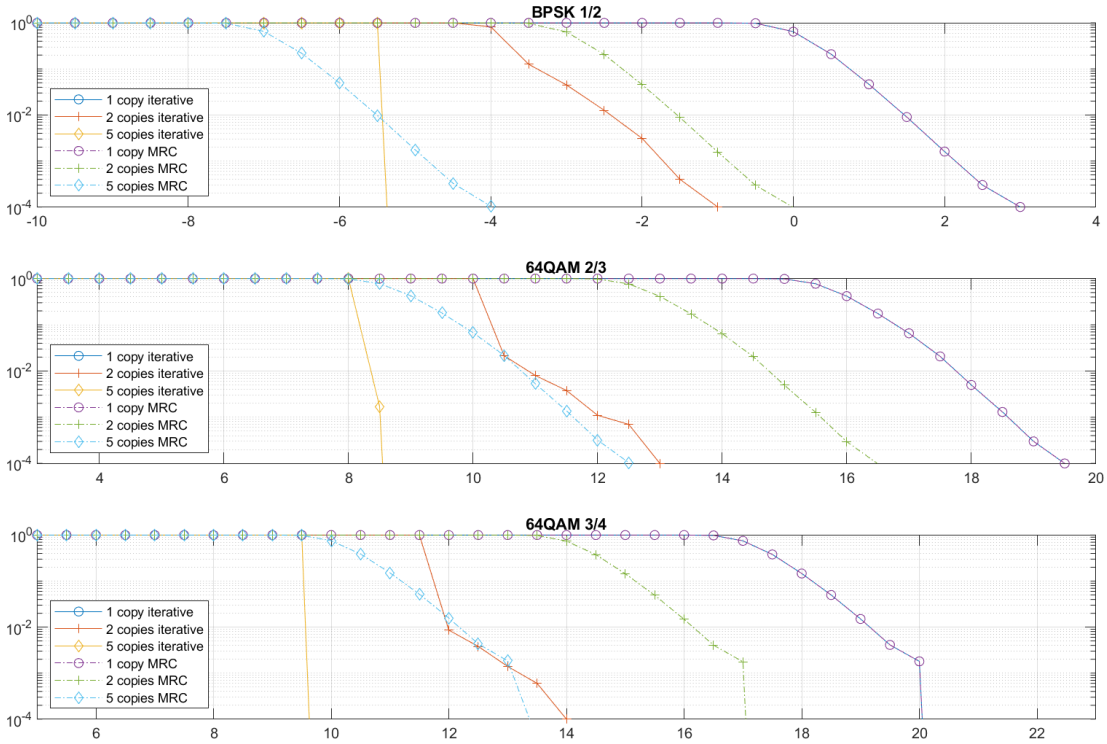

Figure 24. Iterative decoding vs. MRC with 802.11 code over AWGN channel

6.2.2. Simulations over the Rayleigh Channel

This section introduces simulation results of decoding performance with the RESCUE code over the Rayleigh channel. Results can be seen in Figure 36, Figure 37, and Figure 38. Comparison between iterative decoding and MRC can be seen in Figure 39.

From the results, it can be seen that similarly as over the AWGN channel, joint decoding does not provide any additional gain over MRC with a code rate of 1/2. When decoding between two packet copies with a code rate of 2/3, the gain is around 1 dB and 1.5 dB for a rate of 3/4. Although decoding gains are not as high as over the AWGN channel, they can be seen increasing with the number of packet copies.

6.3. Comparison between 802.11 and RESCUE Codes

In this chapter, 802.11 and RESCUE codes are compared against each other in terms of BER and FER. Comparison is made for BPSK 1/2, 64-QAM 2/3, and 64-QAM 3/4, the same code rates used in earlier sections.

6.3.1. Simulations over the Additive White Gaussian Noise Channel

Comparison between RESCUE and 802.11 simulation results over the AWGN channel can be seen in Figure 40, Figure 41, and Figure 42. From the results, it can be seen that in general, the RESCUE code performs better than 802.11. Gain is apparent

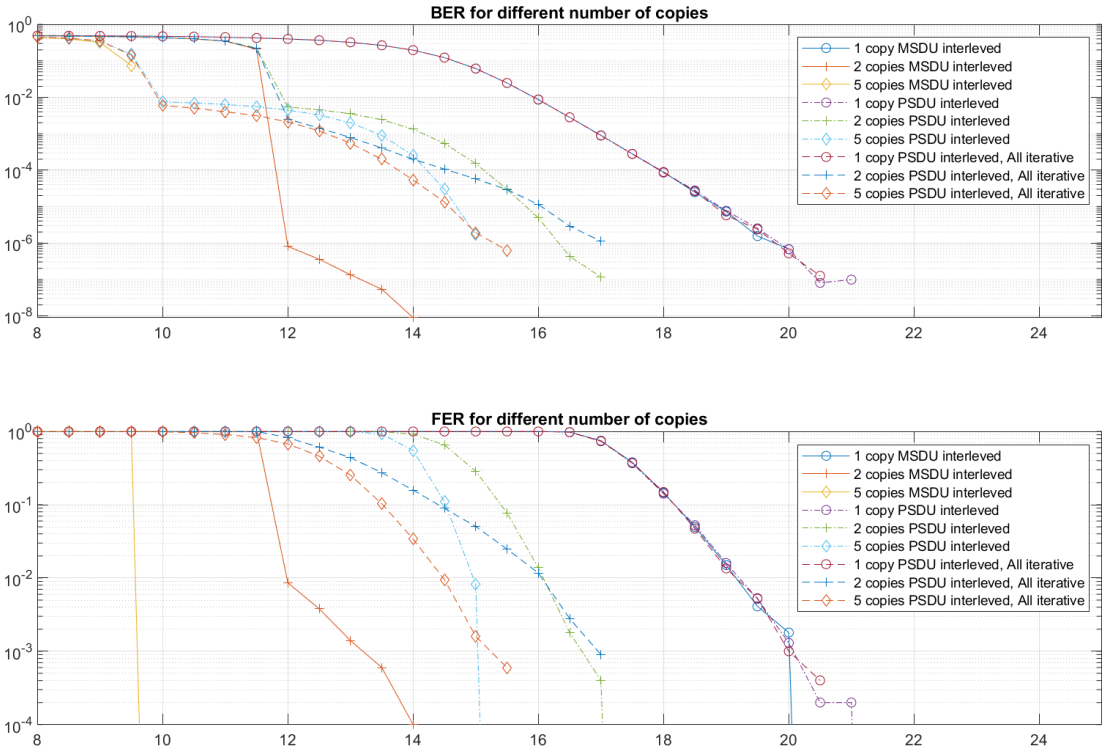

Figure 25. Simulation results for 64-QAM 3/4 with 802.11 code interleaving only MSDU

when decoding only one packet copy as the RESCUE code's turbo effect pushes BER down within a short region. In contrast, the 802.11 code starts converging earlier but reaches low BER later. When decoding two copies, the difference is smaller, but the RESCUE code can be seen to perform slightly better. As for the RESCUE code, the joint decoding gain increases with the number of packets, whereas for the 802.11 it is decreasing. The performance cap between the RESCUE and 802.11 increases with the number of copies.

6.3.2. Simulations over the Rayleigh Channel

Comparison between RESCUE and 802.11 code over the Rayleigh channel can be seen in Figure 43, Figure 44, and Figure 45. For a single copy, the RESCUE code performs better with all the code rates. Although BER figures are similar, the FER performance of the RESCUE code is 0.5 dB - 1 dB better, meaning erroneously decoded packets with the RESCUE code have in average more bit errors than with 802.11. Performance with two copies is almost similar for both codes, except with a code rate of 2/3 where 802.11 outperforms RESCUE around 0.5 dB in terms of FER. However, with five copies, RESCUE is clearly performing better than 802.11, with a code rate of 3/4 around 2 dB better.

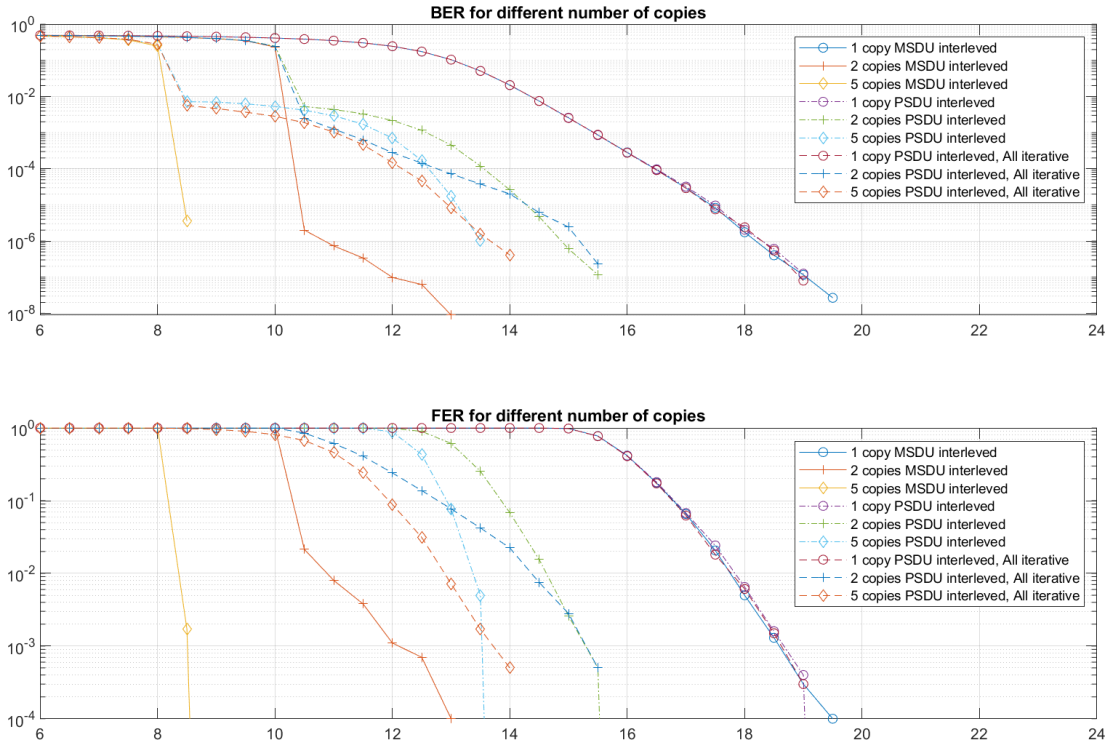

Figure 26. Simulation results for 64-QAM 2/3 with 802.11 code interleaving only MSDU

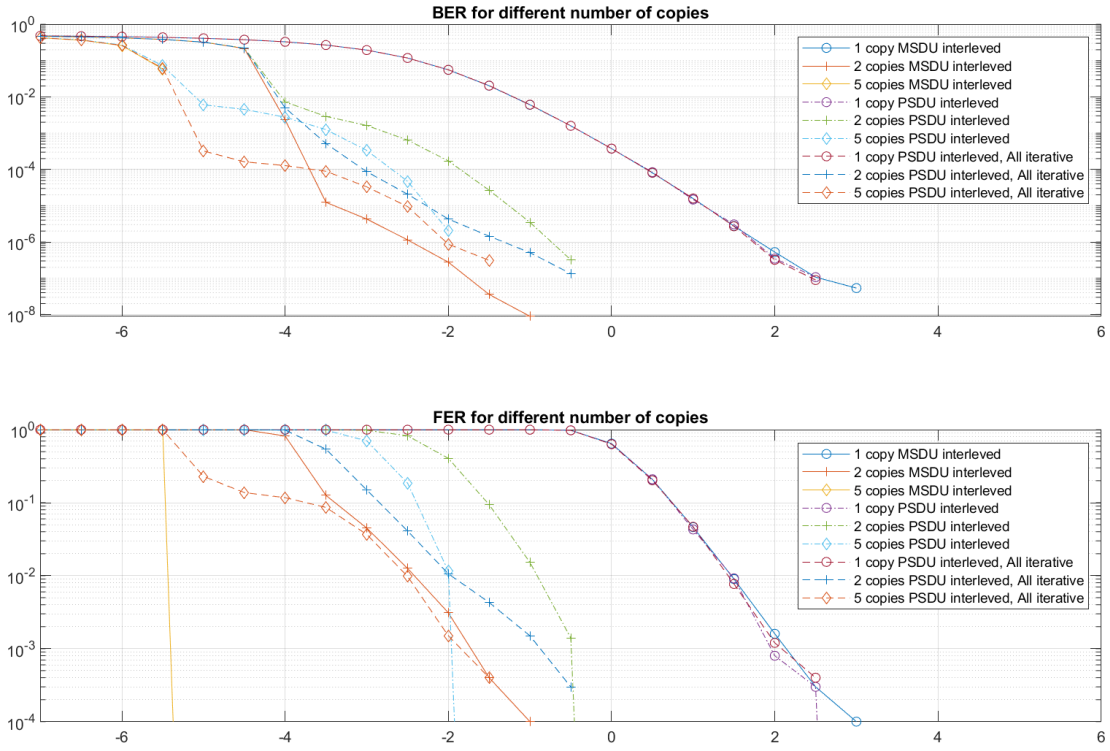

Figure 27. Simulation results for BPSK 1/2 with 802.11 code interleaving only MSDU

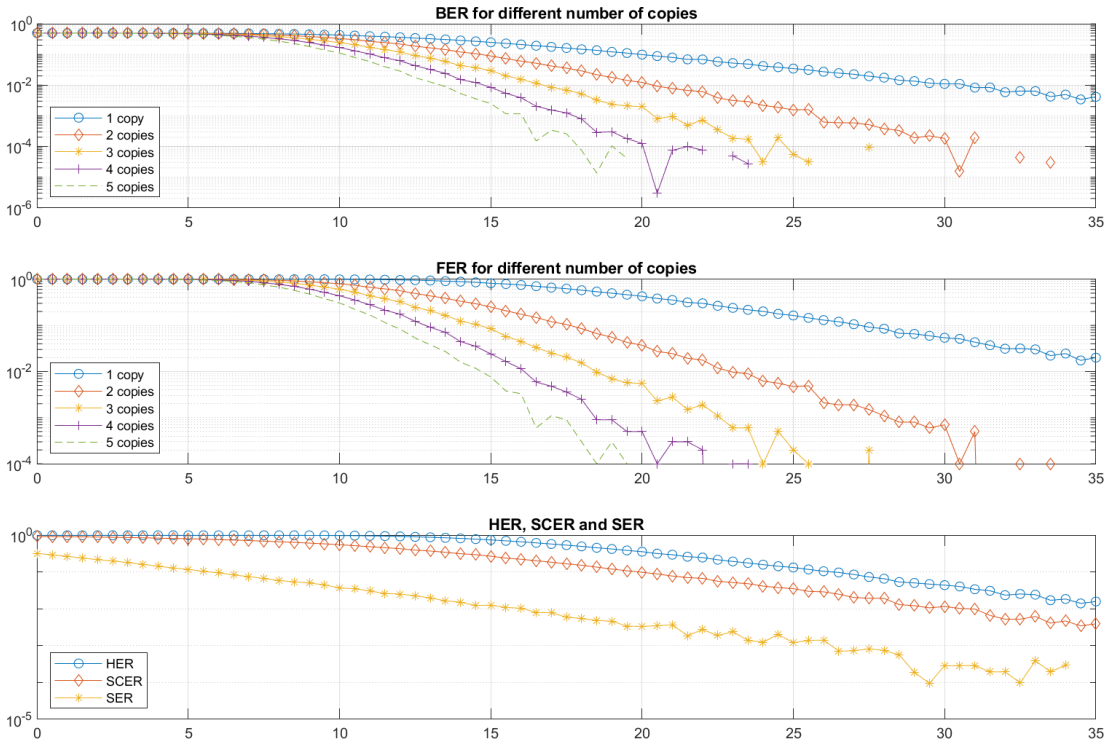

Figure 28. Simulation results for 64-QAM 3/4 with 802.11 code over Rayleigh channel

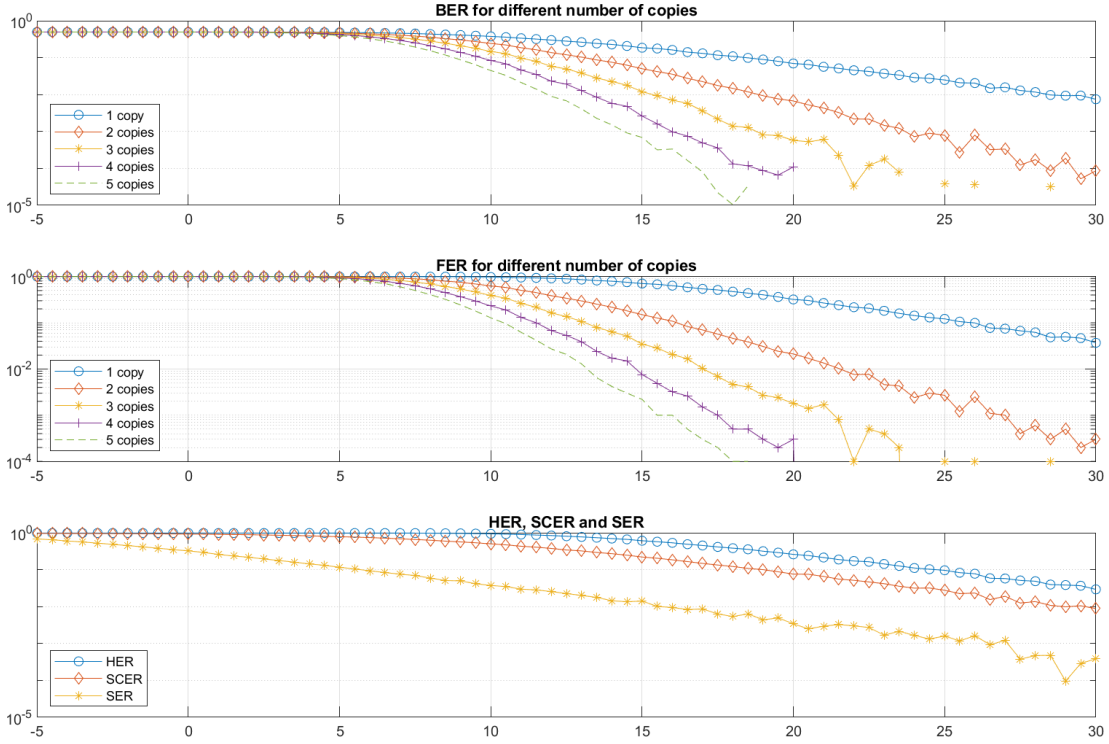

Figure 29. Simulation results for 64-QAM 2/3 with 802.11 code over Rayleigh channel

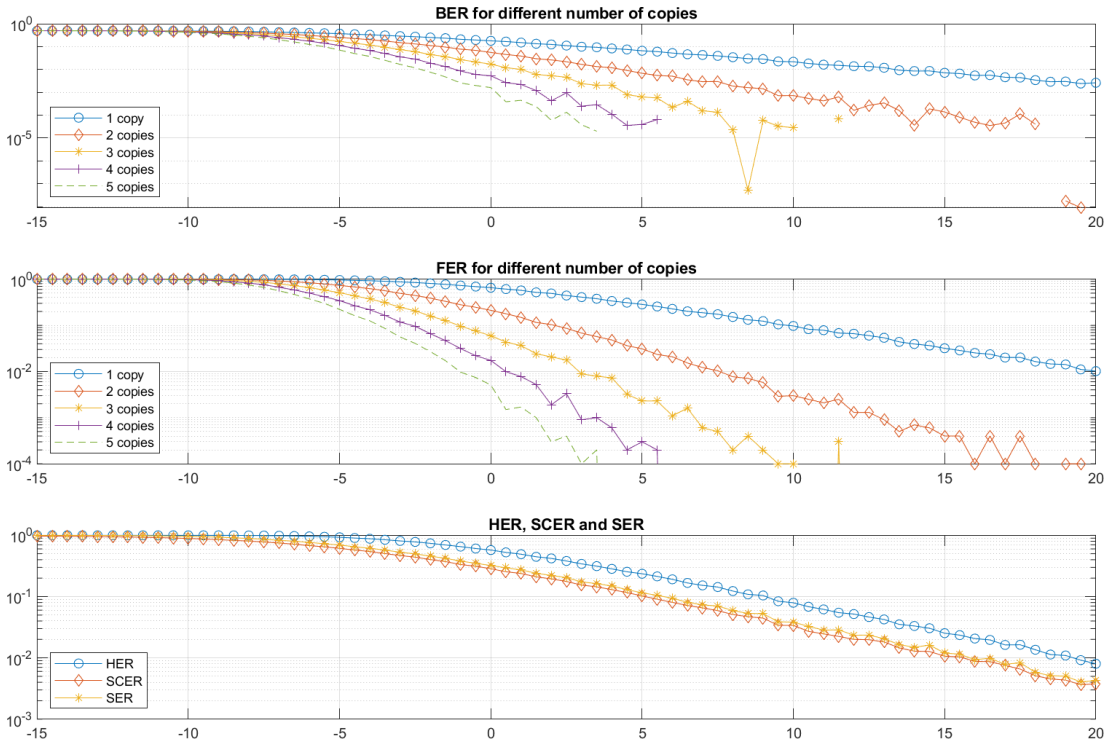

Figure 30. Simulation results for BPSK 1/2 with 802.11 code over Rayleigh channel

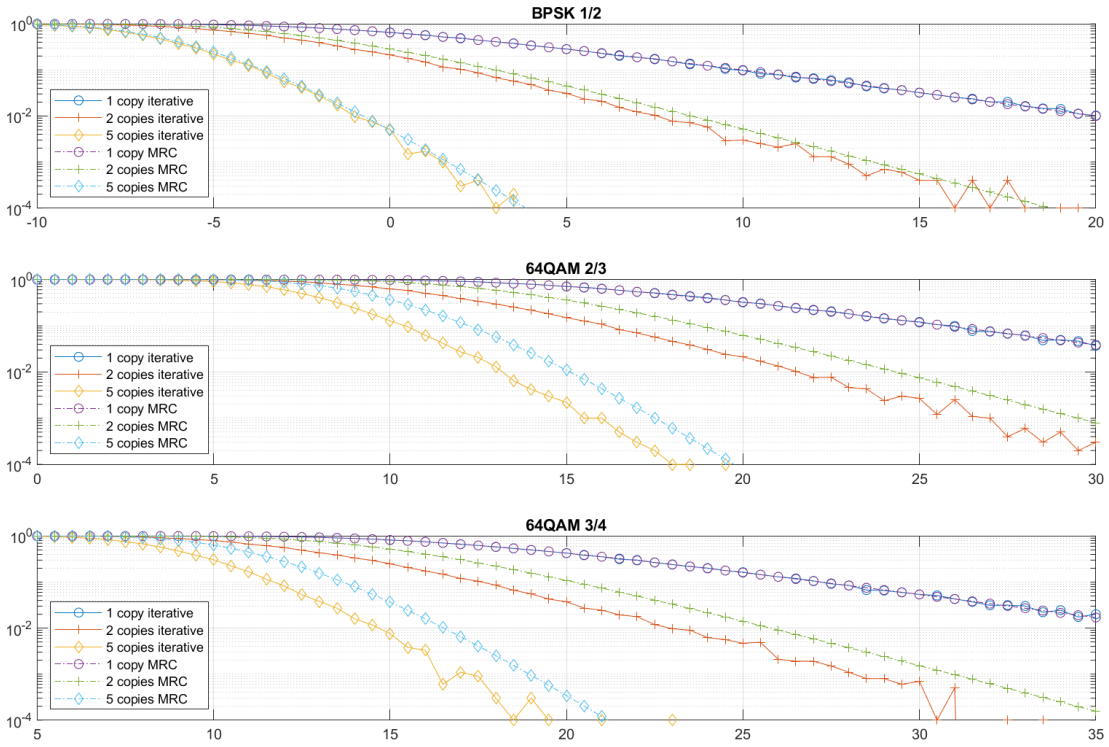

Figure 31. Iterative decoding vs. MRC with 802.11 code over Rayleigh channel

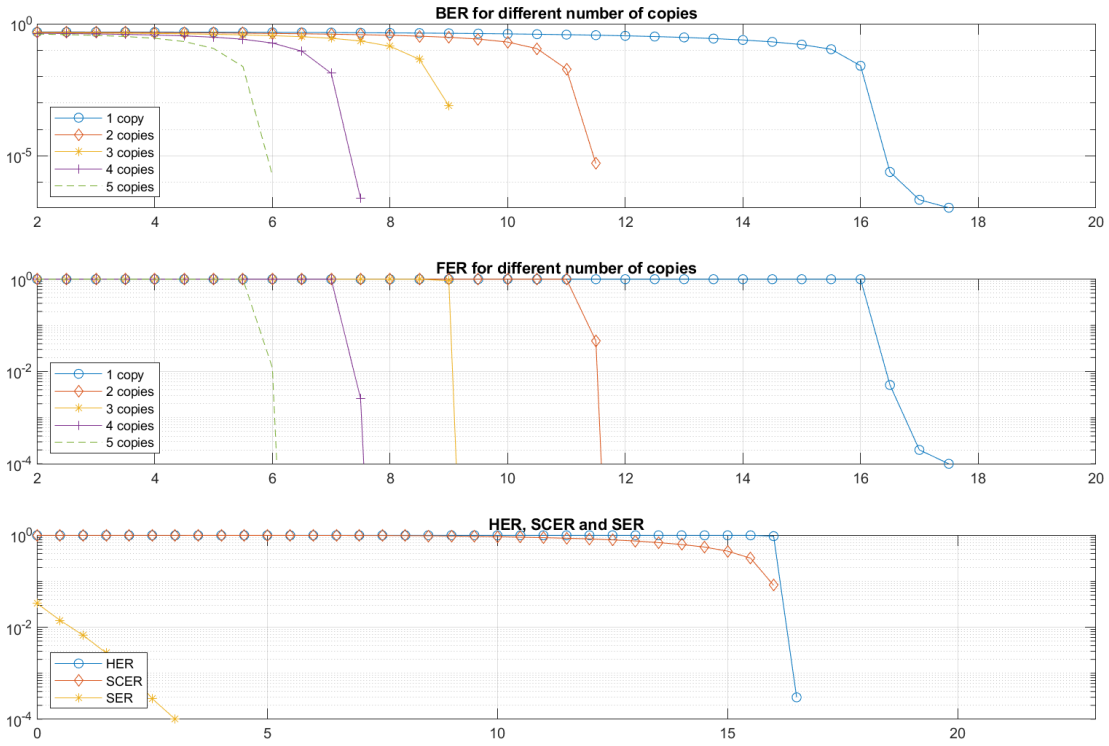

Figure 32. Simulation results for 64-QAM 3/4 with rescue code

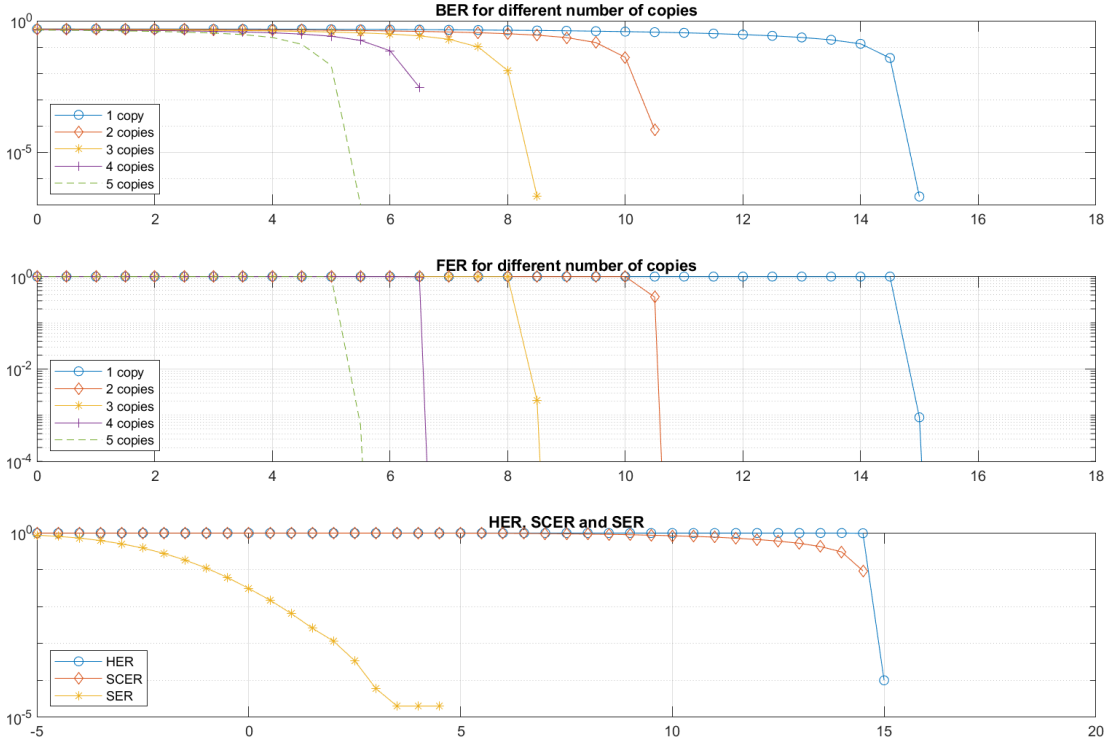

Figure 33. Simulation results for 64-QAM 2/3 with rescue code

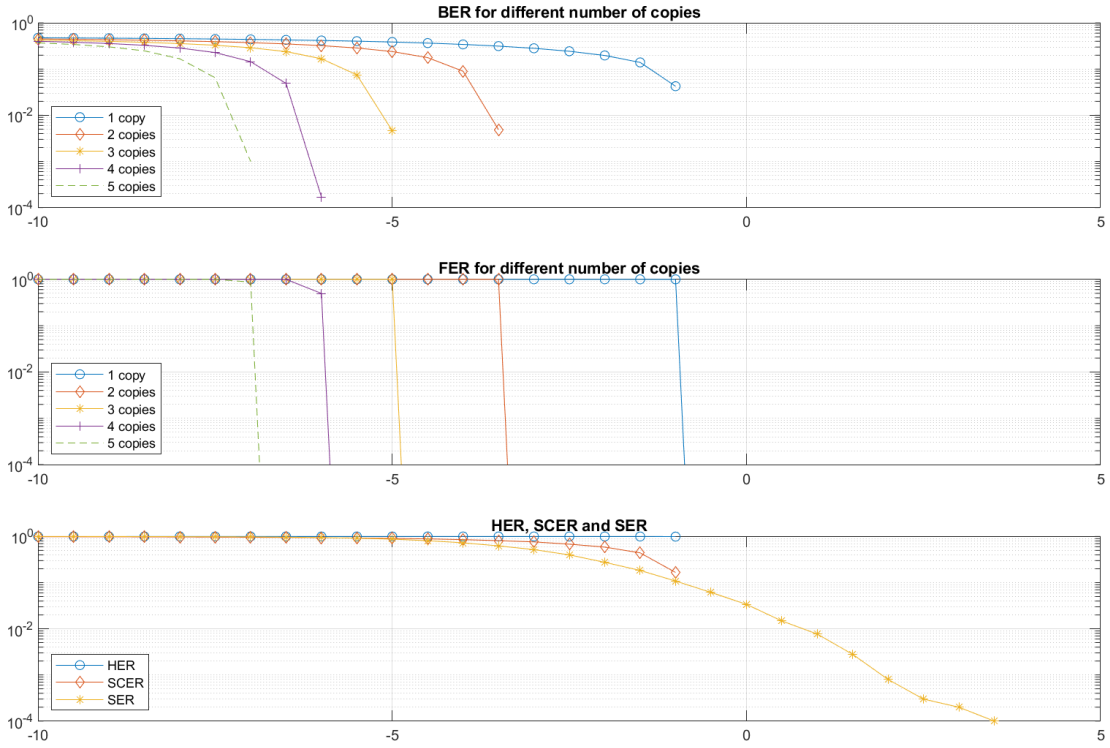

Figure 34. Simulation results for BPSK 1/2 with rescue code

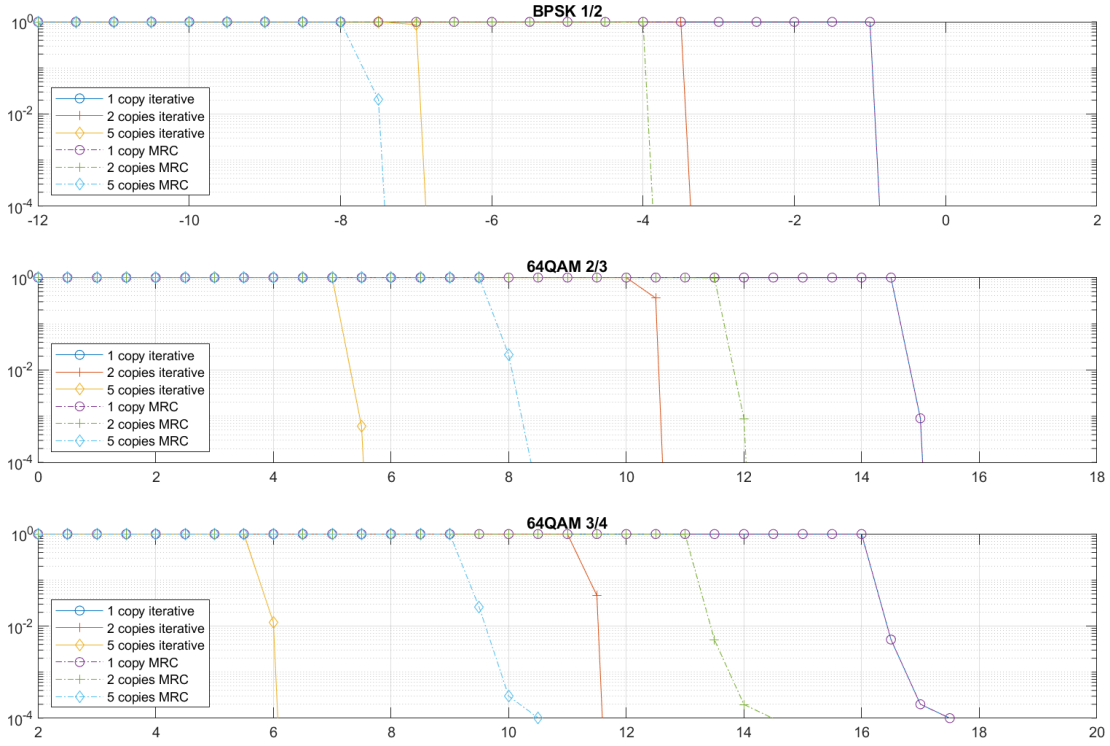

Figure 35. Iterative decoding vs. MRC with RESCUE code over AWGN channel

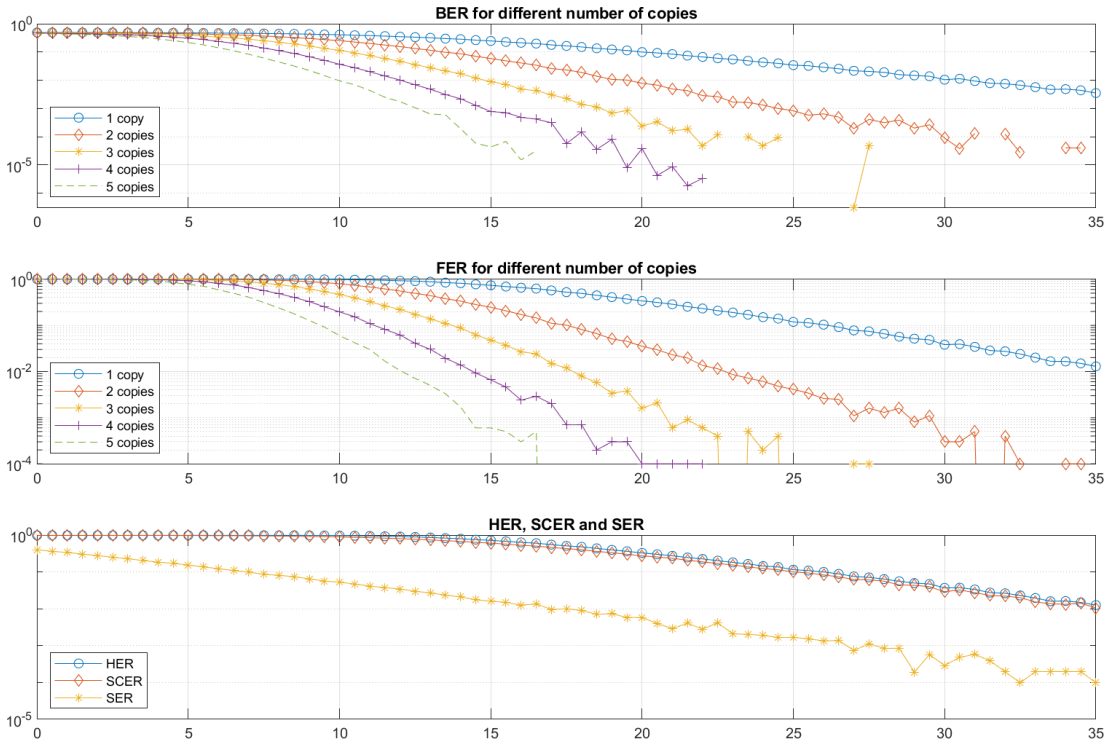

Figure 36. Simulation results for 64-QAM 3/4 with rescue code over Rayleigh channel

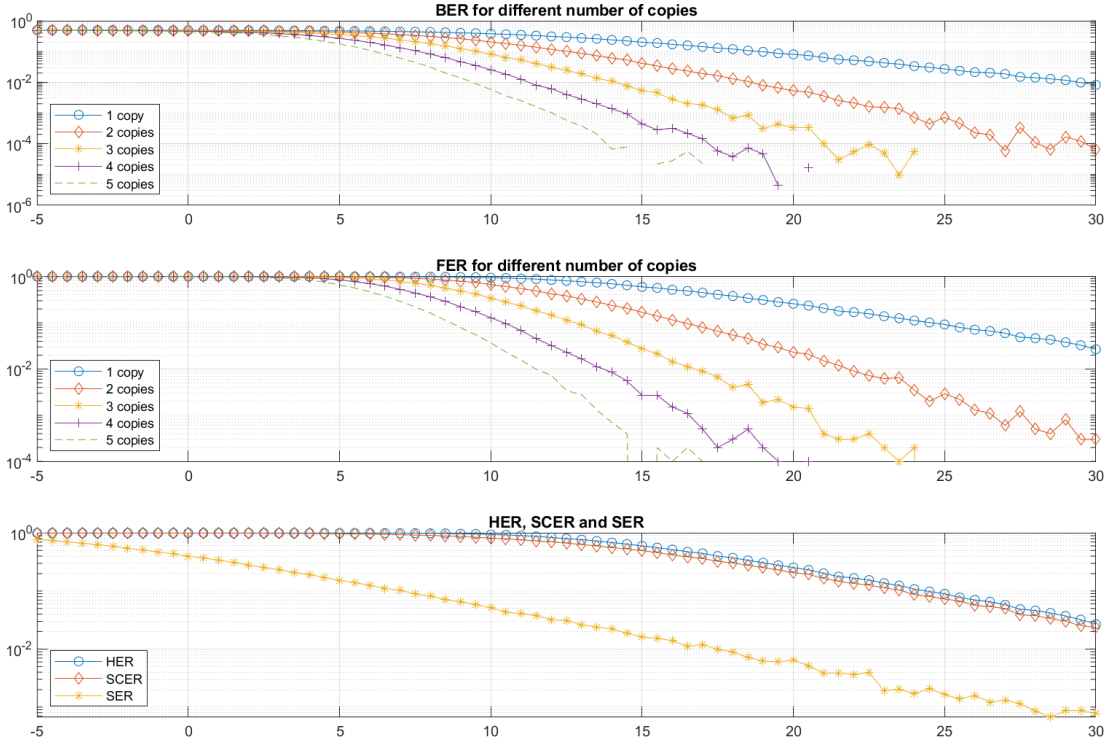

Figure 37. Simulation results for 64-QAM 2/3 with rescue code over Rayleigh channel

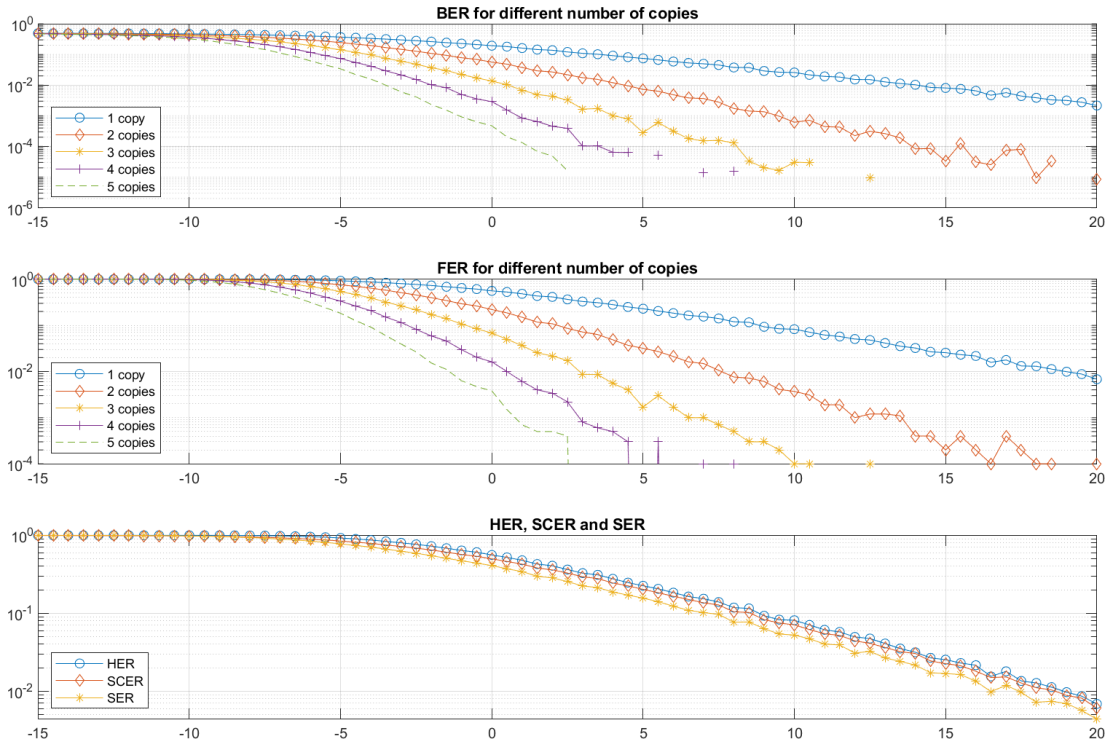

Figure 38. Simulation results for BPSK 1/2 with rescue code over Rayleigh channel

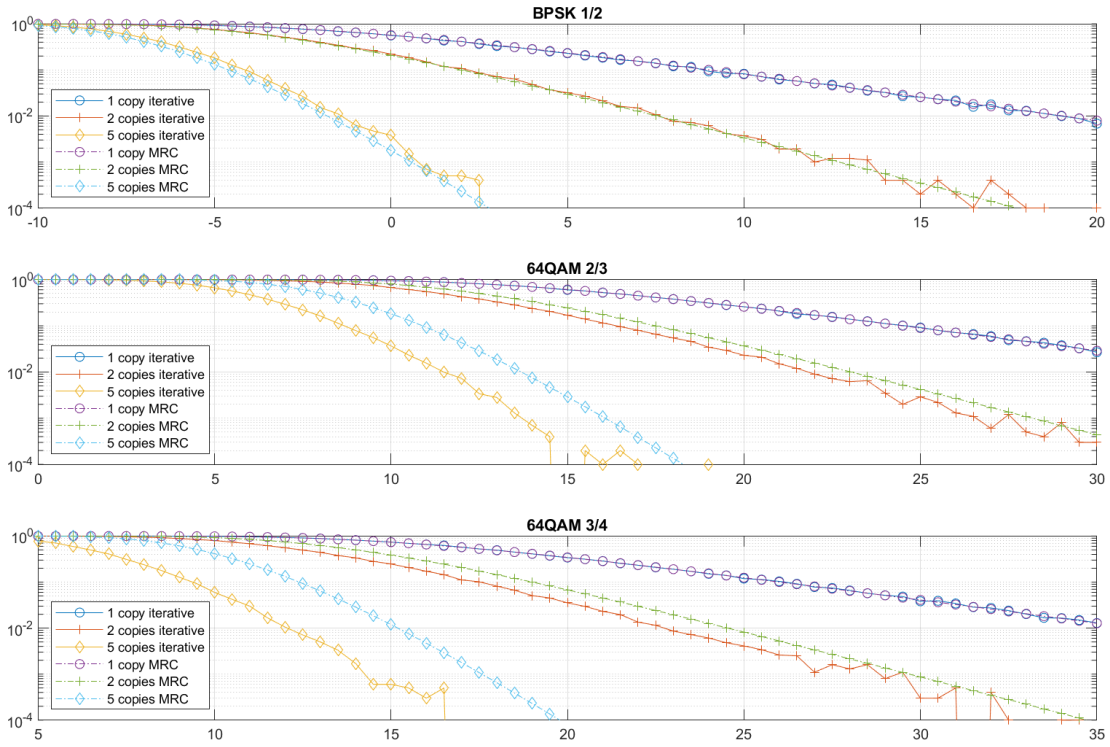

Figure 39. Iterative decoding vs. MRC with RESCUE code over Rayleigh channel

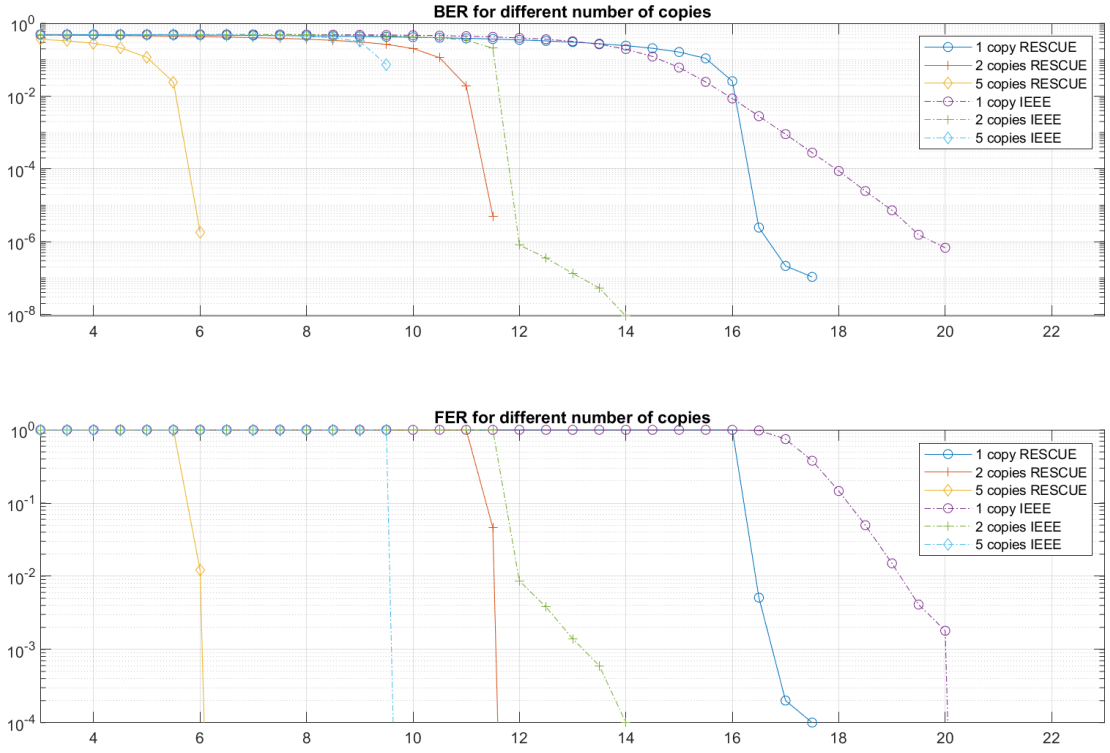

Figure 40. Comparision between RESCUE and 802.11 code for 64-QAM 3/4

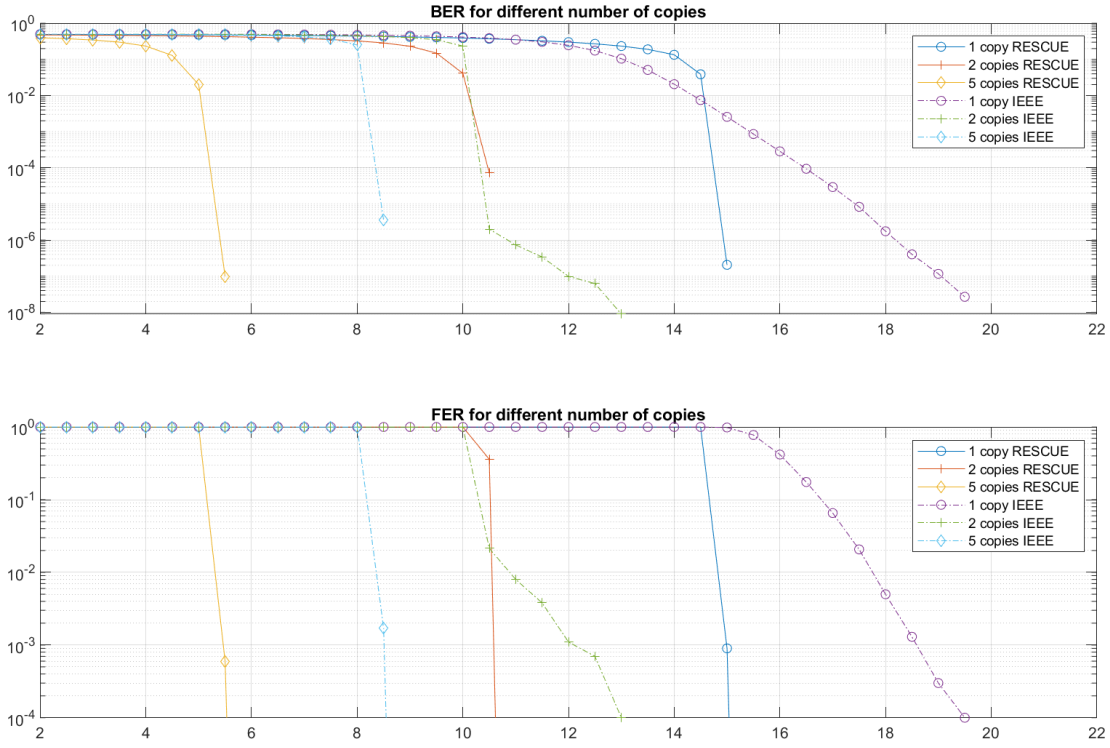

Figure 41. Comparision between RESCUE and 802.11 code for 64-QAM 2/3

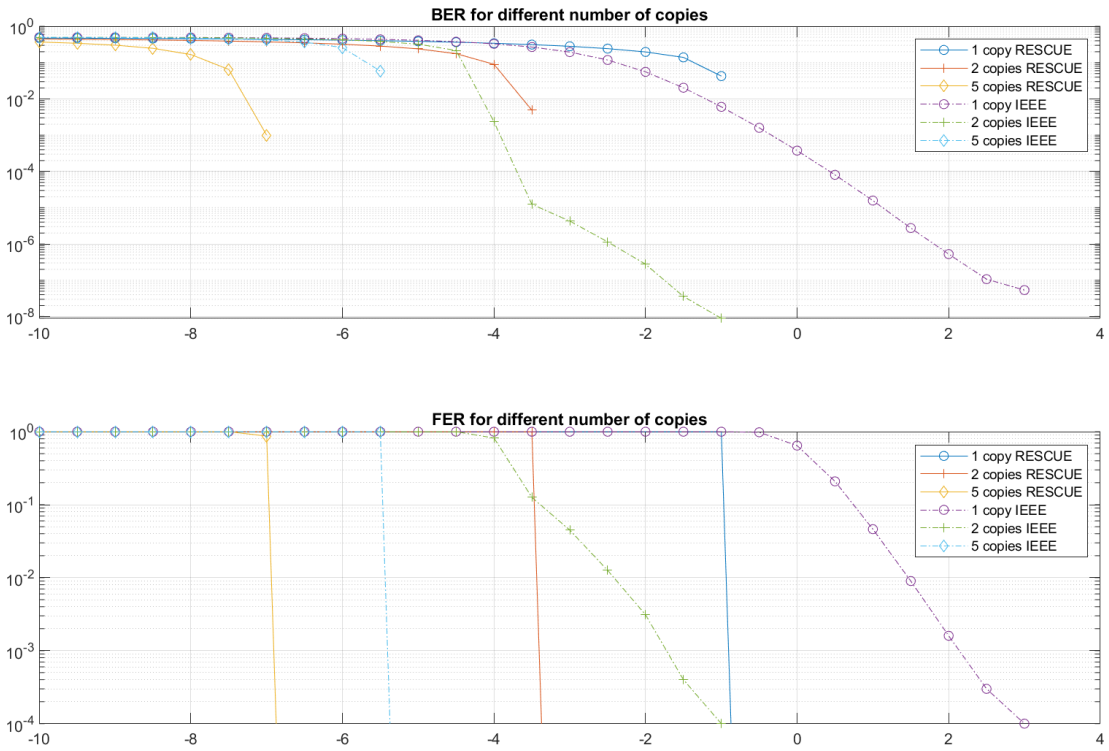

Figure 42. Comparision between RESCUE and 802.11 code for BPSK 1/2

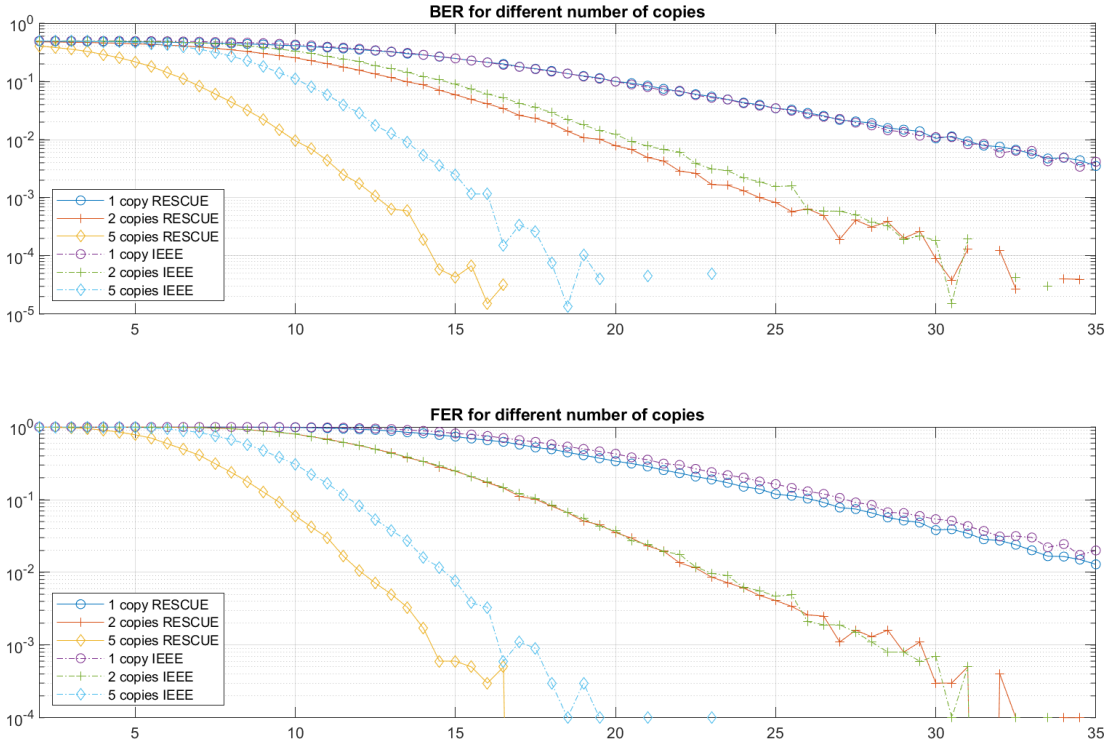

Figure 43. Comparision between RESCUE and 802.11 code for 64-QAM 3/4 over Rayleigh channel

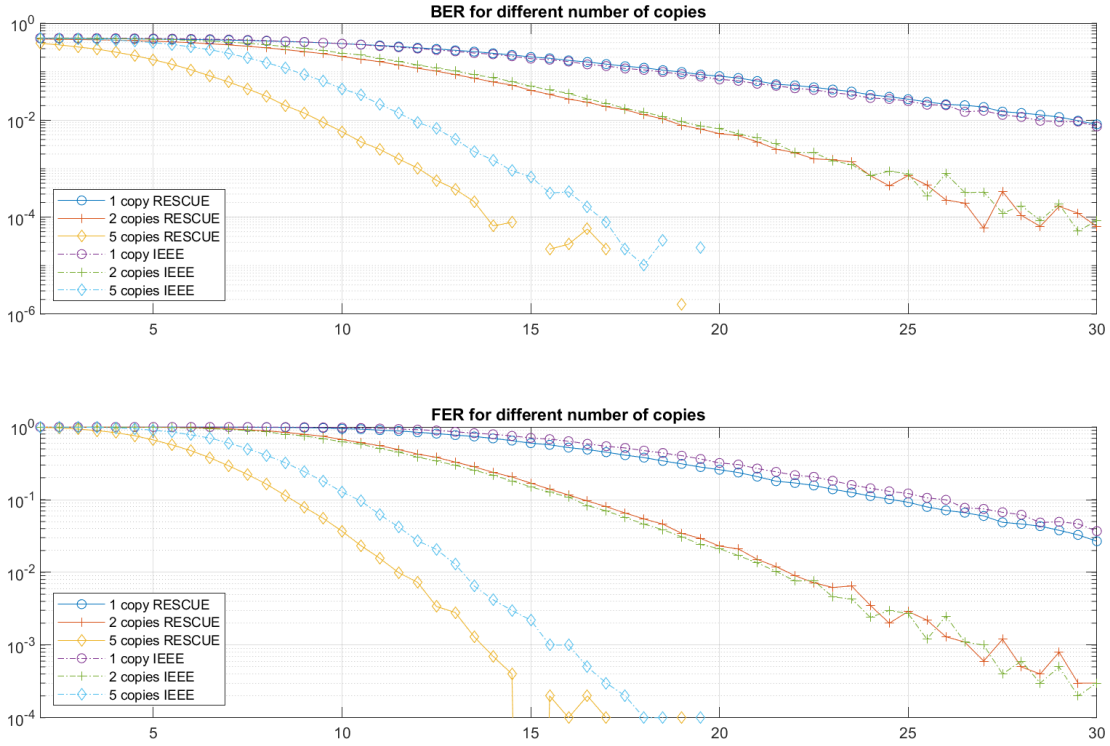

Figure 44. Comparision between RESCUE and 802.11 code for 64-QAM 2/3 over Rayleigh channel

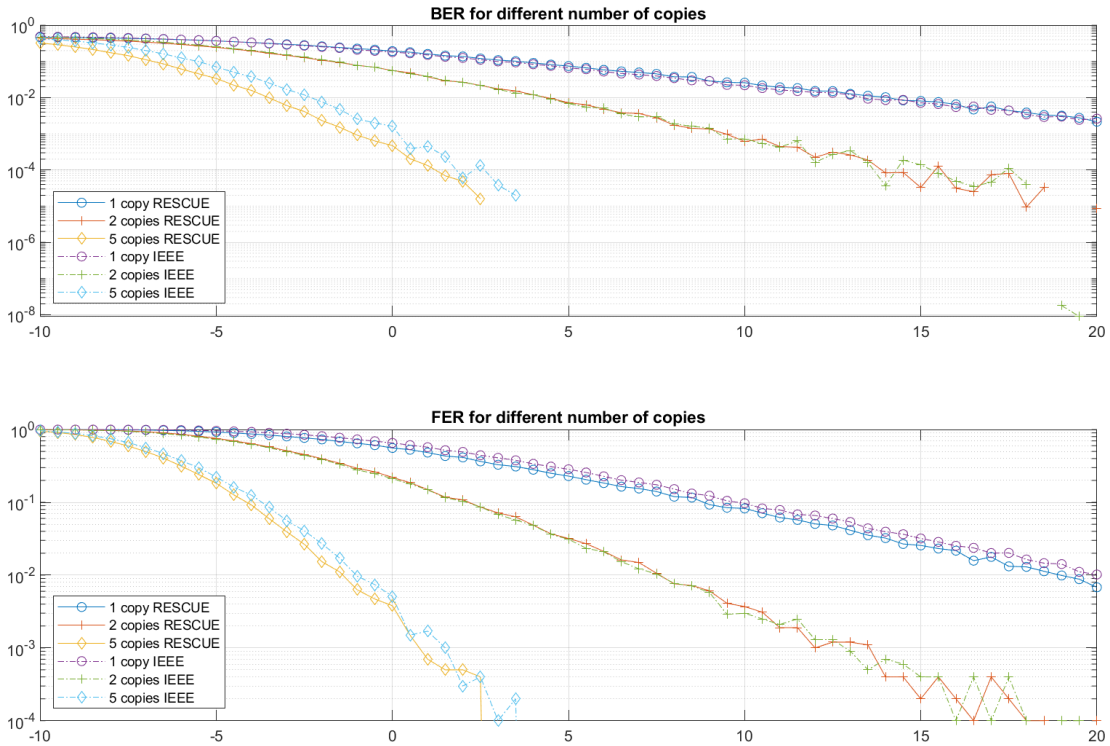

Figure 45. Comparision between RESCUE and 802.11 code for BPSK 1/2 over Rayleigh channel

7. DISCUSSION

In this thesis performance of RESCUE code applied on top of 802.11 physical layer has been compared against 802.11 decoding in scenarios where multiple packet copies have been sent over AWGN or Rayleigh channel. Decoding has been done in an iterative manner applying Log-MAP decoding, which allows taking soft input and gives soft output allowing easy integration with other decoders. For both codes also EXIT analysis has been done to understand better code properties.

To investigate realistic scenarios revealing real-life problems in proposed technology, only minimal changes were made on the 802.11 physical layer to allow iterative decoding. These changes are mainly the addition of an extra interleaver for MSDU/PSDU and different transmission order of information bits in case of interleaved PSDU. Furthermore, adding the interleaver is made so that it does not introduce any additional signaling overhead as the scrambler index is used simultaneously as an interleaver index.

Some of the technical decisions have been taken without any deeper analysis, and any deeper performance analysis is left for future work. One of these decisions is encoding the signal field, which in this thesis is encoded only by outer code because of its small block size that may greatly reduce the performance of the turbo decoding. As other possible options were not evaluated, no bigger role has been given for signal field decoding when interpreting results.

The proposed technology can be seen as HARQ, where the transmitter sends more coded bits until decoding is successful. Another well-known technology based on sending more coded bits is called incremental redundancy. The main difference between proposed technology and incremental redundancy is that in incremental redundancy, more redundancy bits are sent, which are then combined with original bits without separate iterative process[7], whereas in the proposed technology, turbo-like iteration is used between copies. When incremental redundancy is applied on convolutional codes, it may cause some loss of performance with the first transmission as it places requirements for code puncturing that may lead to non-optimal patterns. To allow easy sending of redundancy bits, puncturing patterns are designed so that punctured bits in lower code rates are also punctured in higher code rates. The technology proposed in this thesis does not carry such limitations but comes with the price of added interleaver and higher decoding complexity. Added interleaver, on the other hand, may improve performance against non-white noise sources like partial collision with another transmission as it spreads errors over the whole packet. Adding incremental redundancy support for 802.11 has been investigated earlier in [3].

Simulation results in Chapter 6 show good performance for payload decoding when the whole PSDU is interleaved. In this case the original transmission order, where the header is at the beginning and the FCS at the end, is broken as the interleaver will spread these bits over the frame. This would lead to non-backward compatibility with status quo standards. Therefore, the proposed technology is not fully feasible for wider use. In the case of 802.11 code is used, backward compatibility can be guaranteed easily by sending the first copy without extra interleaving. Hence in this case using scrambler index as interleaver index would not work as it would require to

[7]There can be iterations in the decoding process itself for example with LDPC codes but no separate iteration between new and old data.

reset scrambler index for each new packet making the scrambler more or less useless. This means that some other criteria to select interleaver would be needed. At simplest this could be just the copy number. These options are not investigated any further in the context of this thesis and are left for future research. Interleaving only MSDU is seen to be a sub-optimal solution due to errors in the header and the FCS parts. In such case, better protection (lower code rate) for header and FCS would be needed.

Comparison of payload decoding performance for a single copy between 802.11 and RESCUE code shows that the RESCUE code performs better in most of the cases. Notable is that due to the small memory size of the RESCUE code (inner code 1 bit, outer code 2 bits) also decoding complexity of the RESCUE code is lower than the 802.11 code (having memory of 6 bits) even with 8 inner iterations used with the RESCUE code in this thesis as decoding complexity using BCJR is relative to 2^M, where M is a memory of the code. Although the 802.11 code start converging at lower SNR than RESCUE this is not visible in FER figures as even a single bit error may drop the whole frame. When SNR gets over the RESCUE code's turbo cliff it starts clearly outperforming the 802.11 code in both, FER and BER figures. In this thesis, some evidence of the RESCUE BER floor is only seen with the highest code rate of 3/4. Most likely some BER floor exists also for lower rates but 10000 packets used in simulations per SNR points is not enough to show it.

Payload decoding performance when multiple copies are sent over the AWGN channel the RESCUE code is performing better for all the simulated numbers of copies although with two copies difference in performance is not big and at lower SNR the 802.11 may perform slightly better. A notable difference between the RESCUE and the 802.11 codes is the behavior when the number of copies is increasing: For the RESCUE code gain between iterative decoding vs. MRC is increasing with a number of copies whereas for the 802.11 trend is opposite. The clear benefit for the RESCUE code in iterative decoding between packet copies is that it can be pushed over the turbo cliff using *a priori* information as seen in EXIT analysis.

Similar performance favoring the RESCUE code is seen as well with the Rayleigh channel although with two copies performance is almost identical. In this thesis channel gain for the Rayleigh channel is constant over the single transmission of a packet copy, which is the case only in a relatively fast fading environment where link adaptation is not able to adapt into a changing environment leading in many cases usage of non-optimal MCS for transmission. Although such fast fading may not be a common case for 802.11 like systems it is a good measure for decoding performance in case of uneven SNR of packet copies.

The main objective of this thesis is to investigate the performance of the proposed technology for range extension at a low SNR regime and possible usages at higher SNR to improve performance. Based on the simulations decoding performance of the PSDU would allow 2.7 times longer range in free space with BPSK 1/2 MCS using 5 packet copies and the RESCUE code and 1.3 times by only changing the 802.11 code to the RESCUE at FER target of 10^{-3}. Even though the decoding performance of the PSDU is good enough for such gains there are severe limiting factors seen in simulations. The main limiting factors are signal decoding, scrambler seed estimation, and header errors. As all of these need to be successfully decoded in order to detect the packet, simulations are showing that these errors are almost entirely killing the gain from the iterative decoding of multiple copies. The signal carrying the MCS

and length information, which might be possible to estimate from the received data although might be challenging. An easier option would be adding better protection for the signal as a lower code rate. Signal errors are mainly an issue at lower MCS as it is always encoded with BPSK 1/2, and therefore detection rarely fails when SNR is high. The header is needed to know if the packet is for the receiver or not as it carries to address information. One option for header detection working on lower SNR would be some correlation-based criteria, although it might cause false positives in case of a low threshold. Scrambler seed errors are particularly problematic with the proposed technology as it has a two-fold purpose, scrambler seed as in vanilla 802.11 and the interleaver index. This issue could be overcome by decoupling scrambler seed from the interleaver as proposed earlier in this chapter and by using the same scrambler seed for all the copies of a packet to allow estimating it from a larger data set. These possible improvements are left for future research and not further covered in this thesis.

A possible use case of this technology at a high SNR regime would be re-sending erroneously decoded packets with higher MCS, which would be jointly decoded with the original copy. This would possibly allow keeping MCS higher for the given SNR due to a smaller penalty of erroneous transmission, but this is as well left for future research and not analyzed here. However, similar limitations are also seen at higher SNR except for signal errors, and therefore, similar corrections could be used to mitigate them as at low SNR.

Even if all these listed limiting factors discovered in simulations would be fixed, a low SNR region is problematic for the receiver from the channel estimation and synchronization point of view. These issues are not visible in simulations as this thesis assumes perfect synchronization meaning the channel is flat and known and no disparities between transmitter and receiver. Also, packet detection from the training sequence becomes problematic at low SNR as correlation thresholds need to be low, causing lots of false-positive detections.

8. SUMMARY

The purpose of this thesis has been to investigate the usage of the RESCUE code as part of 802.11, joint decoding between 802.11 packet copies with and without RESCUE code, and possible use cases of the proposed technology for range extension with low SNR and as a HARQ technology. In this thesis, a detailed comparison between the 802.11 and RESCUE code has been made in several scenarios, including EXIT analysis and simulations modeling point-to-point wireless links with AWGN and Rayleigh channels.

In simulations, three different physical layer implementations have been benchmarked against each other for a different number of packet copies: plain 802.11, 802.11 physical layer with RESCUE code keeping header and FCS on their original place, and 802.11 physical layer with RESCUE code and whole PSDU interleaved.

Both EXIT analysis and simulations show the superiority of the RESCUE code for payload decoding against 802.11 with similar complexity except for the case with two packet copies where there is no big difference between codes. Also, it is seen that both codes are showing under joint decoding better performance than MRC except the scenario with the RESCUE code keeping header and FCS on their original places. The results with the AWGN channel show that the RESCUE code performs around 3dB better for a single copy and around 1.5 dB - 3.5 dB better with five copies depending on MCS at FER 10^{-3}.

Simulation results show good performance for payload decoding but also reveal severe issues with the proposed technology: At low SNR, signal decoding errors would drop most of the packets. At high SNR, where signal decoding normally succeeds, header and scrambler decoding errors are causing similar issues. For these reasons, the proposed technology is not usable as it and would require further development. However, when these errors are ignored, and only payload decoding is taken into account, the proposed technology works well, especially with the RESCUE code and would be usable for both rate extension at low SNR and as HARQ technology at high SNR.

9. REFERENCES

[1] (2016) IEEE standard for information technology-telecommunications and information exchange between systems local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012) , pp. 1–3534.

[2] Anwar K. & Matsumoto T. (2012) Accumulator-assisted distributed turbo codes for relay systems exploiting source-relay correlation. IEEE Communications Letters 16, pp. 1114–1117.

[3] Strinati E., Simoens S. & Boutros J. (2003) Performance evaluation of some hybrid ARQ schemes in IEEE 802.11a networks. In: IEEE Semiannual Vehicular Technology Conference, vol. 4, pp. 2735–2739.

[4] He J., He X., Hou J., Jiang W., Kuuhlmorgen S., Matsumoto T., Paatelma A., Qian S., Tervo V., Wolf A. & Yiu N. (2016) Final report of the coding/decoding algorithms and performance comparison. Tech. rep.

[5] Hagenauer J. & Hoeher P. (1989) A Viterbi algorithm with soft-decision outputs and its applications. In: IEEE Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond', vol. 3, pp. 1680–1686.

[6] Robertson P., Villebrun E. & Hoeher P. (1995) A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. In: IEEE International Conference on Communications, vol. 2, pp. 1009–1013.

[7] Breiling M. & Hanzo L. (1997) Optimum non-iterative turbo-decoding. In: IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, vol. 2, pp. 714–718.

[8] ten Brink S., Speidel J. & Yan R.H. (1998) Iterative demapping and decoding for multilevel modulation. In: IEEE Global Communications Conference, vol. 1, pp. 579–584.

[9] Gray F. (U.S. Patent 2 632 058, Nov. 1947), Pulse code communication.

[10] Shannon C.E. (1948) A mathematical theory of communication. The Bell System Technical Journal 27, pp. 379–423.

[11] Fano R.M. (1949) The transmission of information. Tech. rep., USA.

[12] Golay M.J.E. (1949) Notes on digital coding. In: Proc. IRE. 37: 657.

[13] Hamming R.W. (1950) Error detecting and error correcting codes. The Bell System Technical Journal 29, pp. 147–160.

[14] Reed I.S. & Solomon G. (1960) Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8, pp. 300–304. URL: https://doi.org/10.1137/0108018.

[15] Elias P. (1955) Coding for noisy channels. IRE Convention Record 3 .

[16] MacKay D.J.C. & Neal R.M. (1995) Good codes based on very sparse matrices. In: C. Boyd (ed.) Cryptography and Coding, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 100–111.

[17] Berrou C. (France Patent 267 597 1B1, Apr. 1991), Corrective error coding method with at least two systemic convolutive codes in parallel, iterative decoding method, corresponding decoding module and decoder.

[18] Berrou C., Glavieux A. & Thitimajshima P. (1993) Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In: IEEE International Conference on Communications, vol. 2, pp. 1064–1070.

[19] Arikan E. (2009) Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. IEEE Transactions on Information Theory 55, pp. 3051–3073.

[20] Goldsmith A. (2005) Wireless Communications. Cambridge University Press.

[21] Jin H. & McEliece R.J. (2000) General coding theorems for turbo-like codes. In: IEEE International Symposium on Information Theory, pp. 120–.

[22] Garello R., Chiaraluce F., Pierleoni P., Scaloni M. & Benedetto S. (2001) On error floor and free distance of turbo codes. In: IEEE International Conference on Communications, vol. 1, pp. 45–49.

[23] Shannon C. (1956) The zero error capacity of a noisy channel. IRE Transactions on Information Theory 2, pp. 8–19.

[24] Comroe R. & Costello D. (1984) ARQ schemes for data transmission in mobile radio systems. IEEE Journal on Selected Areas in Communications 2, pp. 472–481.

[25] Bahl L., Cocke J., Jelinek F. & Raviv J. (1974) Optimal decoding of linear codes for minimizing symbol error rate (corresp.). IEEE Transactions on Information Theory 20, pp. 284–287.

[26] Woddard J. & Hanzo L. (2005) Turbo Convolutional Coding, "John Wiley & Sons, Ltd", chap. 5. pp. 105–171. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/047085474X.ch5.

[27] Andrews K., Heegard C. & Kozen D. (1997) A theory of interleavers. Tech. rep., USA.

[28] Dolinar S. & Divsalar D. (1995) Weight distributions for turbo codes using random and nonrandom permutations, tda progress report. Tech. rep., USA.

[29] Hagenauer J., Offer E. & Papke L. (1996) Iterative decoding of binary block and convolutional codes. IEEE Transactions on Information Theory 42, pp. 429–445.

[30] Robertson P. (1994) Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes. In: IEEE Global Communications Conference, vol. 3, pp. 1298–1303.

[31] 3GPP (2020) Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding. Technical Specification (TS) 36.212, 3rd Generation Partnership Project (3GPP). URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2426, version 16.0.0.

[32] ten Brink S. (1999) Convergence of iterative decoding. Electronics Letters 35, pp. 806–808.

[33] ten Brink S. (2001) Convergence behavior of iteratively decoded parallel concatenated codes. IEEE Transactions on Communications 49, pp. 1727–1737.

[34] Hagenauer J. & Hoeher P. (1989) Concatenated Viterbi decoding. In: Proceedings of the Fourth Joint Swedish-Soviet International Workshop on Information Theory, Studentlitteratur Lund, pp. 29–33.

[35] Wiberg N. (2001) Codes and Decoding on General Graphs. Ph.D. thesis, Linköping Studies in Science and Technology, Department of Electrical Engineering Linköping University, S-581 83 Linköping, Sweden.

[36] Brännström F. (2004) Convergence analysis and design of multiple concatenated codes. Ph.D. thesis, Chalmers Tekniska Hogskola, Chalmers University of Technology, S-412 96 Goteborg, Sweden.

[37] Brännström F., Rasmussen L.K. & Grant A.J. (2005) Convergence analysis and optimal scheduling for multiple concatenated codes. IEEE Transactions on Information Theory 51, pp. 3354–3364.

[38] Schneider C., Skoblikov O., Lorenz M., Hollmach N., Käske M., Natkaniec M., Sikora M., Kosek-Szott K., Szott S., Wszołek J., Gozdecki J., Loziak K., Prasnal L., Sośnik S., Trzeciakowski L., Adigun O., Pavlatos N., Khalife H., Seddar J., Tervo V., Paatelma A., He J. & Jokinen M. (2016) Final report on practical assessment of the RESCUE architecture. Tech. rep.

[39] Paatelma A., Nguyen D.H., Saarnisaari H., Kandasamy N. & Dandekar K.R. (2017) Reinforcement learning system to mitigate small-cell interference through directionality. In: IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications, pp. 1–7.

[40] Yuen J.H. (1983) Deep Space Telecommunications Systems Engineering. Springer Publishing Company, Incorporated, 1st ed.

[41] Gilhousen K., Heller J., Jacobs I. & Viterbi A. (1971) Coding systems study for high data rate telemetry links. Tech. rep., USA.

[42] gr-ieee802-11 (accessed 2.1.2020). URL: https://github.com/bastibl/gr-ieee802-11.

[43] Tse D. & Viswanath P. (2005) Fundamentals of Wireless Communication. Cambridge University Press, 1st ed.